



TÀI LIỆU THÍ NGHIỆM VI XỬ LÝ

BỘ MÔN ĐIỆN TỬ - ĐH BÁCH KHOA TP.HCM

Contents

CHƯƠNG 1	TỔNG QUAN VỀ BỘ THÍ NGHIỆM.....	5
1.1	TỔ CHỨC TÀI LIỆU HƯỚNG DẪN	5
1.2	TỔNG QUAN KIT THÍ NGHIỆM.....	6
1.2.1	Kết nối tín hiệu trên kit thí nghiệm.....	7
1.2.2	Khối testpoint.....	7
1.3	SỬ DỤNG KIT THÍ NGHIỆM.....	7
1.3.1	Cấu hình kit thí nghiệm và kết nối máy tính.....	7
1.4	VIẾT CHƯƠNG TRÌNH VỚI MICROCHIP STUDIO	8
1.5	Sử dụng các chân port lập trình ISP và JTAG	19
1.5.1	Giao diện lập trình ISP:.....	19
1.5.2	Giao diện gõ rối JTAG:	19
CHƯƠNG 2	NÚT NHẤN VÀ LED ĐƠN.....	21
2.1	LÝ THUYẾT CƠ BẢN.....	21
2.2	THIẾT KẾ PHẦN CỨNG.....	21
2.2.1	Khối nút nhấn đơn.....	21
2.2.2	Khối Dip Switch	22
2.2.3	Khối LED đơn.....	23
2.2.4	Khối BAR LED.....	23
2.2.5	Khối thanh ghi dịch.....	24
2.3	LẬP TRÌNH GIAO TIẾP LED VÀ SWITCH.....	25
2.4	LẬP TRÌNH TẠO ĐỘ TRỄ DÙNG CÂU LỆNH	26
2.5	RUNG PHÍM VÀ CHỐNG RUNG	29
2.6	LẬP TRÌNH GIAO TIẾP KHỐI THANH GHI DỊCH.....	30
CHƯƠNG 3	BÀN PHÍM MA TRẬN	33
3.1	LÝ THUYẾT CƠ BẢN.....	33
3.2	GIAO TIẾP BÀN PHÍM MA TRẬN	33
3.3	SỬ DỤNG KHỐI CỔNG AND ĐỂ TẠO NGẮT KHI CÓ PHÍM NHẤN	36
CHƯƠNG 4	HIỂN THỊ DÙNG LED 7 ĐOẠN.....	40
4.1	LÝ THUYẾT CƠ BẢN.....	40
4.2	THIẾT KẾ PHẦN CỨNG.....	41
4.3	HIỂN THỊ LÊN LED 7 ĐOẠN.....	42
4.3.1	Hiển thị lên 1 LED:.....	42
4.3.2	Hiển thị đồng thời lên các LED 7 đoạn	45
4.4	QUÉT LED SỬ DỤNG NGẮT TIMER	46
CHƯƠNG 5	LED MA TRẬN.....	50

5.1	LÝ THUYẾT CƠ BẢN.....	50
5.2	THIẾT KẾ PHẦN CỨNG.....	51
5.3	LẬP TRÌNH GIAO TIẾP LED MA TRẬN.....	51
5.3.1	Kết nối phần cứng trên kit	52
5.3.2	Lập trình hiển thị lên LED ma trận.....	53
CHƯƠNG 6	HIỂN THỊ DÙNG LCD KÝ TỰ.....	57
6.1	LÝ THUYẾT CƠ BẢN.....	57
6.2	THIẾT KẾ PHẦN CỨNG.....	58
6.3	LẬP TRÌNH GIAO TIẾP LCD.....	59
6.4	VIẾT CHƯƠNG TRÌNH HIỂN THỊ KÝ TỰ LÊN LCD	64
CHƯƠNG 7	THÍ NGHIỆM GIAO TIẾP QUA CÔNG NỐI TIẾP	66
7.1	LÝ THUYẾT CƠ BẢN.....	66
7.2	THIẾT KẾ PHẦN CỨNG.....	66
7.3	KẾT NỐI PHẦN CỨNG VÀ CẤU HÌNH PHẦN MỀM	66
7.4	LẬP TRÌNH CHO ATMEGA324 GIAO TIẾP UART	69
CHƯƠNG 8	GIAO TIẾP ADC VÀ KHÓI CẢM BIẾN TƯƠNG TỰ	77
8.1	THIẾT KẾ GIAO TIẾP ADC	77
8.1.1	TỔNG QUAN VỀ ADC TRÊN ATMEGA324PA	77
8.1.2	THIẾT KẾ PHẦN CỨNG	78
8.2	BIẾN TRỞ TẠO ÁP THAY ĐỔI	79
8.2.1	THIẾT KẾ PHẦN CỨNG	79
8.2.2	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH.....	79
8.3	CẢM BIẾN NHIỆT ĐỘ MCP9701	82
8.3.1	THIẾT KẾ PHẦN CỨNG	82
8.3.2	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH.....	82
8.4	CẢM BIẾN ÁNH SÁNG	83
8.4.1	Thiết kế phần cứng.....	83
8.4.2	Kết nối phần cứng và lập trình.....	83
8.5	CẢM BIẾN NHIỆT ĐỘ DS18B20.....	85
8.5.1	THIẾT KẾ PHẦN CỨNG	85
8.5.2	KẾT NỐI VÀ LẬP TRÌNH GIAO TIẾP.....	86
CHƯƠNG 9	BỘ NHỚ EEPROM VÀ SD CARD	Error! Bookmark not defined.
9.1	THIẾT KẾ PHẦN CỨNG KHỐI EEPROM	Error! Bookmark not defined.
9.2	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH	Error! Bookmark not defined.
9.3	THIẾT KẾ PHẦN CỨNG KHỐI SD CARD	Error! Bookmark not defined.
9.4	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH	Error! Bookmark not defined.

CHƯƠNG 10	TOUCH SCREEN LCD	Error! Bookmark not defined.	
10.1	THIẾT KẾ PHẦN CỨNG KHỐI LCD	Error! Bookmark not defined.	
10.2	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH	Error! Bookmark not defined.	
CHƯƠNG 11	ĐỒNG HỒ THỜI GIAN THỰC (RTC).....	Error! Bookmark not defined.	
11.1	THIẾT KẾ PHẦN CỨNG KHỐI RTC	Error! Bookmark not defined.	
11.2	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH	Error! Bookmark not defined.	
CHƯƠNG 12	ĐỘNG CƠ DC.....	87	
12.1	TỔNG QUAN LÝ THUYẾT	87	
12.2	THIẾT KẾ PHẦN CỨNG	89	
	12.3	KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH	89

CHƯƠNG 1 TỔNG QUAN VỀ BỘ THÍ NGHIỆM

1.1 TỔ CHỨC TÀI LIỆU HƯỚNG DẪN

Kit thí nghiệm vi xử lý là bộ thí nghiệm được thiết kế dựa trên họ vi điều khiển AVR. Tài liệu hướng dẫn thí nghiệm này giúp người sử dụng tiếp cận với các kiến thức cơ bản về vi điều khiển AVR nhanh chóng hơn. Tài liệu thí nghiệm bao gồm tài liệu hướng dẫn sử dụng kit thí nghiệm, các bài thí nghiệm, và một số mã nguồn để tham khảo.

Tài liệu hướng dẫn sẽ giới thiệu các thành phần của kit thí nghiệm, được tổ chức thành các phần như sau:

- **Lý thuyết cơ bản:** phần này sẽ tóm tắt sơ lược các kiến thức lý thuyết có liên quan đến bài thí nghiệm.
- **Thiết kế phần cứng:** nội dung của phần này sẽ giúp người sử dụng nắm được chi tiết về sơ đồ và cách thức thiết kế phần cứng của kit thí nghiệm. Người sử dụng cần hiểu rõ các nội dung được đề cập trong phần này.
- **Kết nối phần cứng và lập trình:** phần này sẽ giúp người sử dụng nắm được các kỹ thuật để kết nối các tín hiệu và xây dựng phần mềm đáp ứng yêu cầu của bài thí nghiệm.

Mỗi bài thí nghiệm được tổ chức thành các phần như sau:

- **Mục tiêu:** giúp người học nắm được mục tiêu cụ thể của bài thí nghiệm.
- **Yêu cầu:** phần này sẽ đưa ra yêu cầu cụ thể của bài thí nghiệm.
- **Hướng dẫn:** phần này nêu một số hướng dẫn để SV có thể lập trình dễ dàng hơn
- **Kiểm tra:** giúp người sử dụng đánh giá mức độ đạt được các mục tiêu của bài thí nghiệm, đồng thời gợi ý một số hiệu chỉnh nhằm làm phong phú nội dung thí nghiệm.

Chú ý:

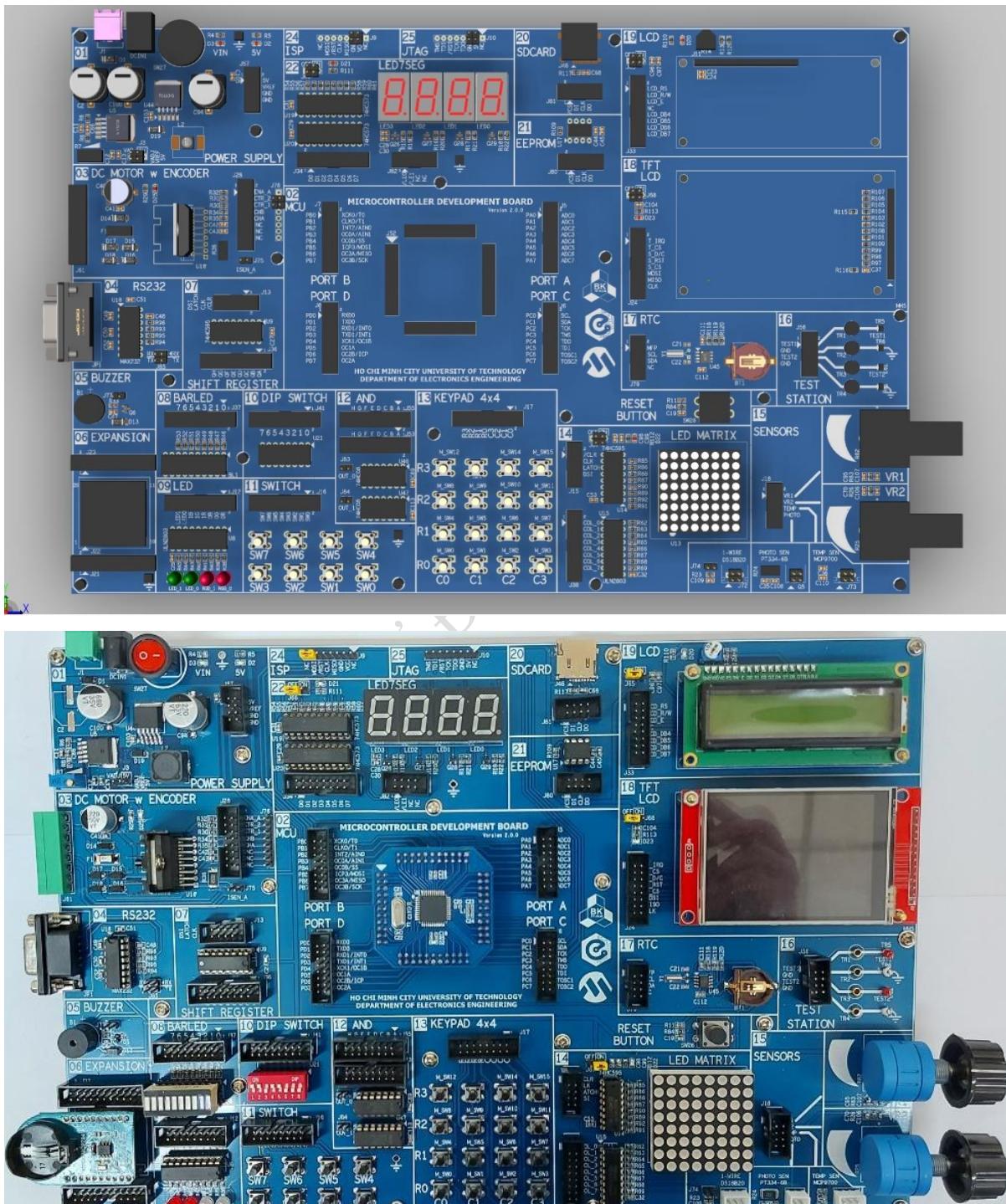
Người học cần xem trước nội dung thí nghiệm và chuẩn bị sẵn chương trình tại nhà để có thể tận dụng tốt thời gian thí nghiệm. Nếu phát hiện có sai sót hay thắc mắc, người học có thể báo trực tiếp Giảng Viên hướng dẫn hoặc email về địa chỉ buiquocbao@hcmut.edu.vn

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

1.2 TỔNG QUAN KIT THÍ NGHIỆM

Kit thí nghiệm có hình dạng và các khối cơ bản như Hình 1. Đi kèm với kit thí nghiệm là bộ dây nối, các cảm biến, động cơ DC, và adaptor nguồn.

Hình 1 Tổng quan kit thí nghiệm vi xử lý



Hình 1 Kit thí nghiệm vi xử lý

1.2.1 Kết nối tín hiệu trên kit thí nghiệm

Bộ thí nghiệm vi xử lý được thiết kế để có thể sử dụng một cách linh hoạt. Xung quanh khối MCU có 4 header sử dụng để kết nối tín hiệu từ 4 PORT A, B, C, D ra các khối ngoại vi. Sinh viên sử dụng các cáp 8x2 hoặc dây pin header để kết nối các tín hiệu.

1.2.2 Khối testpoint

Để dễ dàng sử dụng VOM và oscilloscope đo đặc các tín hiệu, trên kit thí nghiệm cung cấp sẵn các điểm đo. Sinh viên sử dụng dây đơn để kết nối từ tín hiệu cần đo ra các terminal này, và kết nối dây probe từ oscilloscope vào các test point để tiến hành đo đặc.

Lưu ý rằng các test point TR2, TR6, TR4, TR8 đã được kết nối sẵn vào GND. Khi kết nối chú ý tránh kết nối nhầm tín hiệu vào GND.

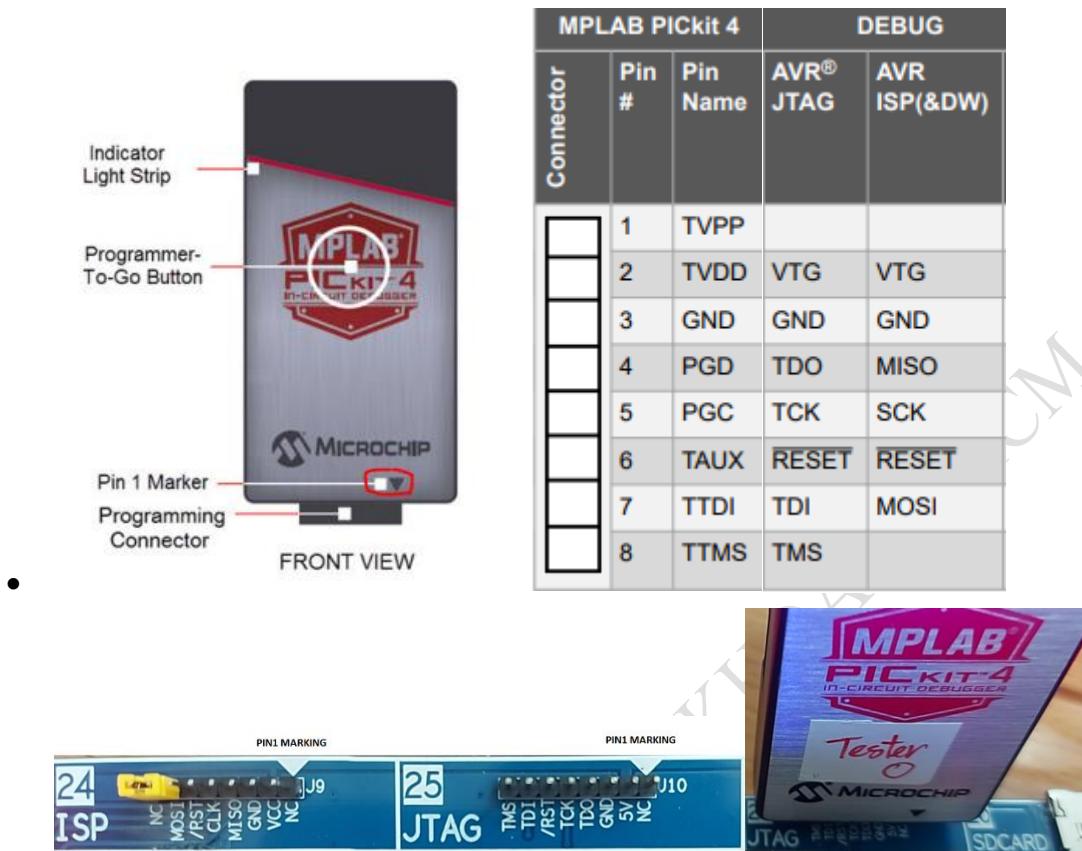


Hình 2 Kết nối và đo đặc tín hiệu

1.3 SỬ DỤNG KIT THÍ NGHIỆM

1.3.1 Cấu hình kit thí nghiệm và kết nối máy tính

- Kết nối bộ PICKIT 4 vào cổng JTAG trên kit thí nghiệm. Nếu muốn sử dụng các chân PC2, PC3, PC4, PC5, ta kết nối vào cổng ISP. Lưu ý chân số 1 của PICKIT sẽ cắm vào chân số 1 của cổng JTAG/ISP.
- Nếu kết nối vào cổng SPI, ta có thể lập trình nhưng không thể gỡ rời (debug) chương trình.
- Kết nối PICKIT4 vào máy tính qua cổng USB
- Kết nối các tín hiệu từ chân port của khối MCU đến các ngoại vi cần thiết.
- Cáp nguồn cho bộ thí nghiệm.
- Bắt đầu tiến trình thí nghiệm.



Hình 3 Kết nối PICKIT 4 vào kit thí nghiệm

1.4 VIẾT CHƯƠNG TRÌNH VỚI MICROCHIP STUDIO

Các hệ thống vi xử lý hoặc vi điều khiển đều cần có một phần mềm (chương trình) để điều khiển hoạt động của nó. Chương trình này được giữ trong bộ nhớ chương trình (program memory) của MCU. Ở cấp thấp nhất, chương trình trong hệ thống là các bit nhị phân thường được gọi là mã máy.

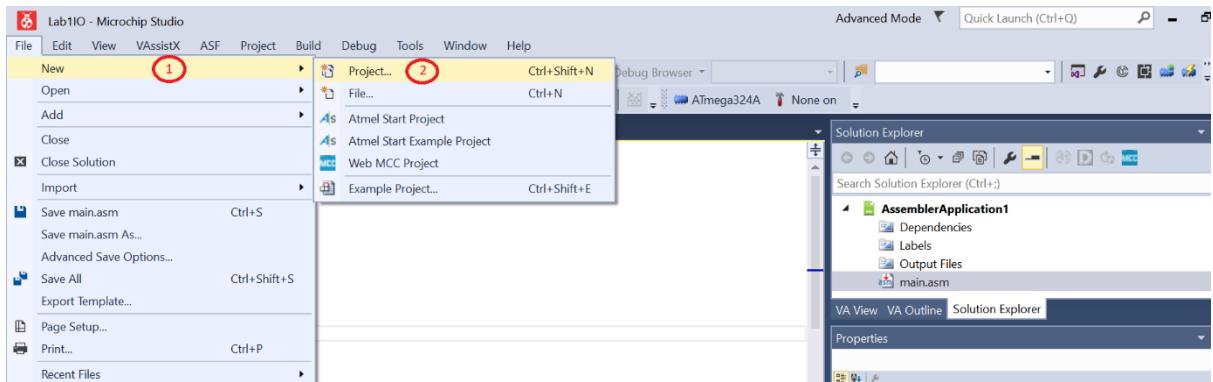
Tuy nhiên, người lập trình rất khó để thao tác với các bit nhị phân. Trong thực tế, các chương trình sẽ được viết trên máy tính bằng hợp ngữ (assembly) hoặc các ngôn ngữ cấp cao khác như C/C++, Basic,... Các chương trình này sẽ cần phải qua bước biên dịch, liên kết để chuyển sang dạng mã máy phù hợp với loại MCU đang dùng. Công cụ để thực hiện các bước này được gọi là chương trình dịch hợp ngữ (assembler), chương trình biên dịch (compiler), và chương trình liên kết (linker). Mỗi loại MCU thường có một chương trình dịch hợp ngữ của riêng nó.

Trong tài liệu thí nghiệm này, người lập trình có thể sử dụng chương trình **Microchip Studio for AVR and SAM Devices**, được download miễn phí trên web của Microchip.

Các bước để tạo một Project:

- Chọn File-New-Project

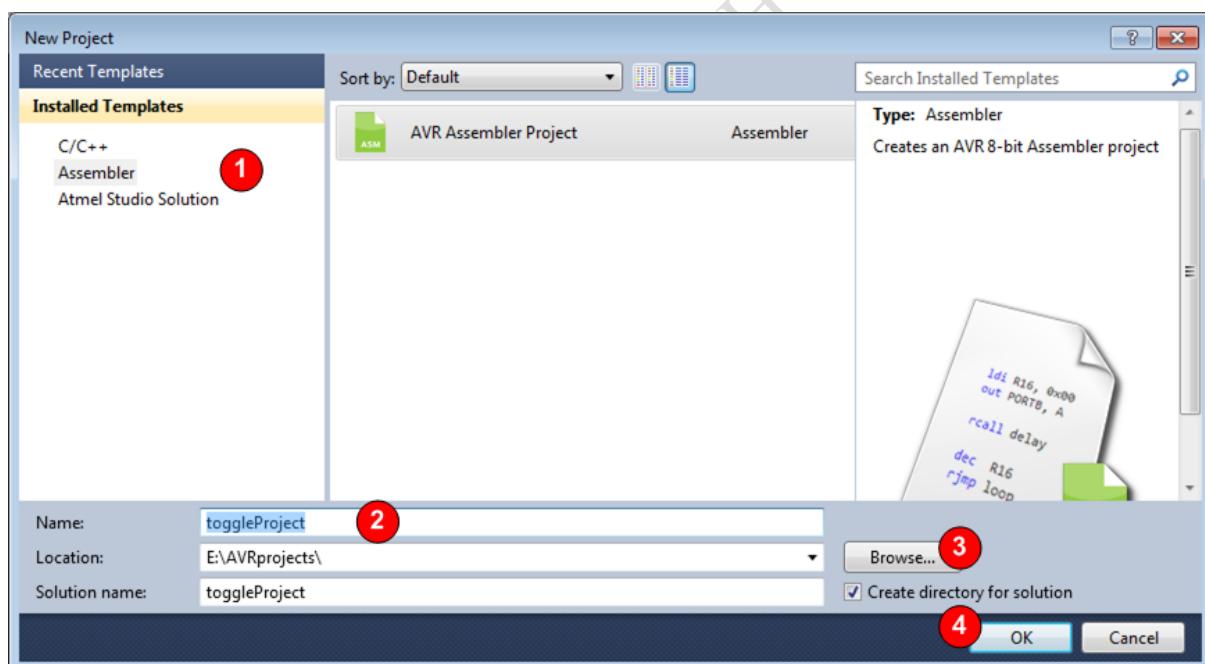
HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ



2. Tạo Project:

Nếu tạo Assembly Project:

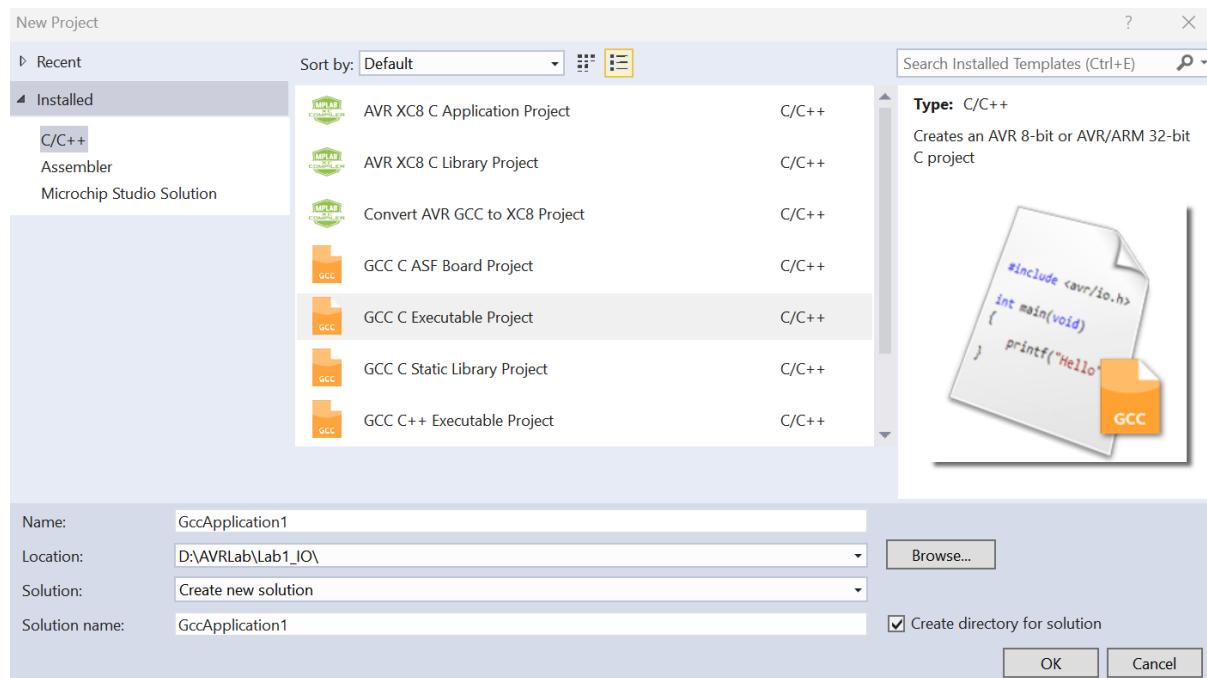
- Chọn AVR Assembler Project
- Đặt tên cho Project
- Chọn một thư mục để chứa Project này
- Nhấn chọn OK



Nếu tạo C Project:

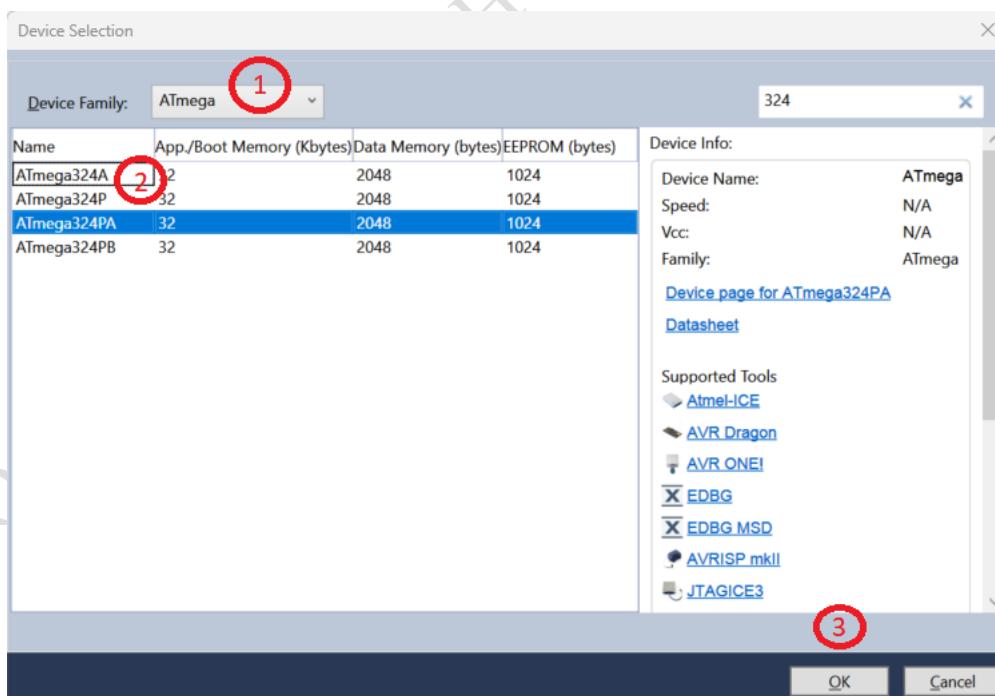
- Chọn C/C++
- Chọn **GCC C Executable Project**
- Đặt tên cho Project
- Chọn một thư mục để chứa Project này
- Nhấn chọn OK

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ



3. Trong phần Device Selection

- Chọn Atmega ở phần Device Family
- Chọn Atmega324PA
- Chọn OK



HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

- Trình biên dịch tự động tạo ra một project với file main.asm. SV soạn thảo file chương trình mới hoặc copy file chương trình vào đây, biên dịch để nạp lên chip

```
main.asm  Data Visualizer

;
; AssemblerApplication1.asm
;
; Created: 2/24/2023 9:38:26 AM
; Author : buiqb
;

; Replace with your application code
LDI R16,0xFF
OUT DDRB,R16
L1: OUT PORTB,R16
LDI R20,0
OUT PORTB,R20
RJMP L1
```

- Biên dịch chương trình

Nhấn F7 hoặc chọn Build-Build Solution. Kết quả biên dịch được thể hiện trong cửa sổ Output. Nếu biên dịch thành công, ta sẽ có file .hex và các file output khác trong thư mục Debug hoặc Release.

```
AssemblerApplication1  main.asm  Data Visualizer
```

```
Output
Show output from: Build
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "D:\AVRLab\Lab1_IO\AssemblerApplication1.asmproj".
Done building target "Build" in project "AssemblerApplication1.asmproj".
Done building project "AssemblerApplication1.asmproj".

Build succeeded.
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ======
```

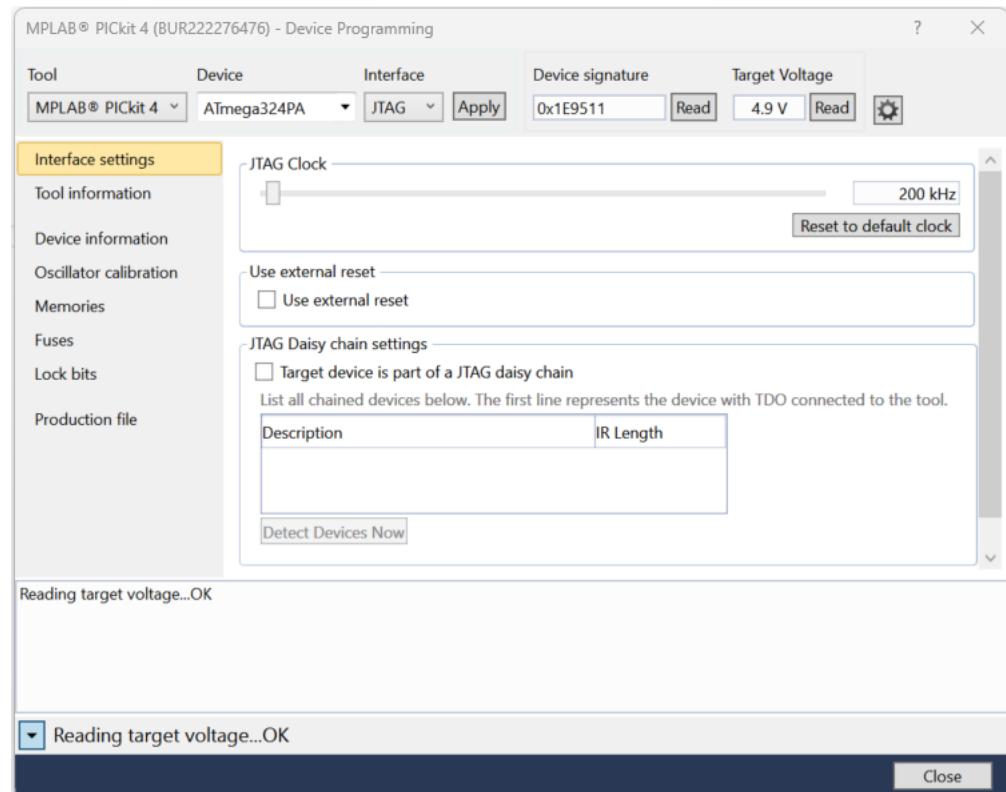
- Nạp chương trình

Ta có thể nạp chương trình khi kết nối PICKIT vào kit thí nghiệm bằng giao tiếp ISP hoặc JTAG

Chọn Tool-Device Programming.

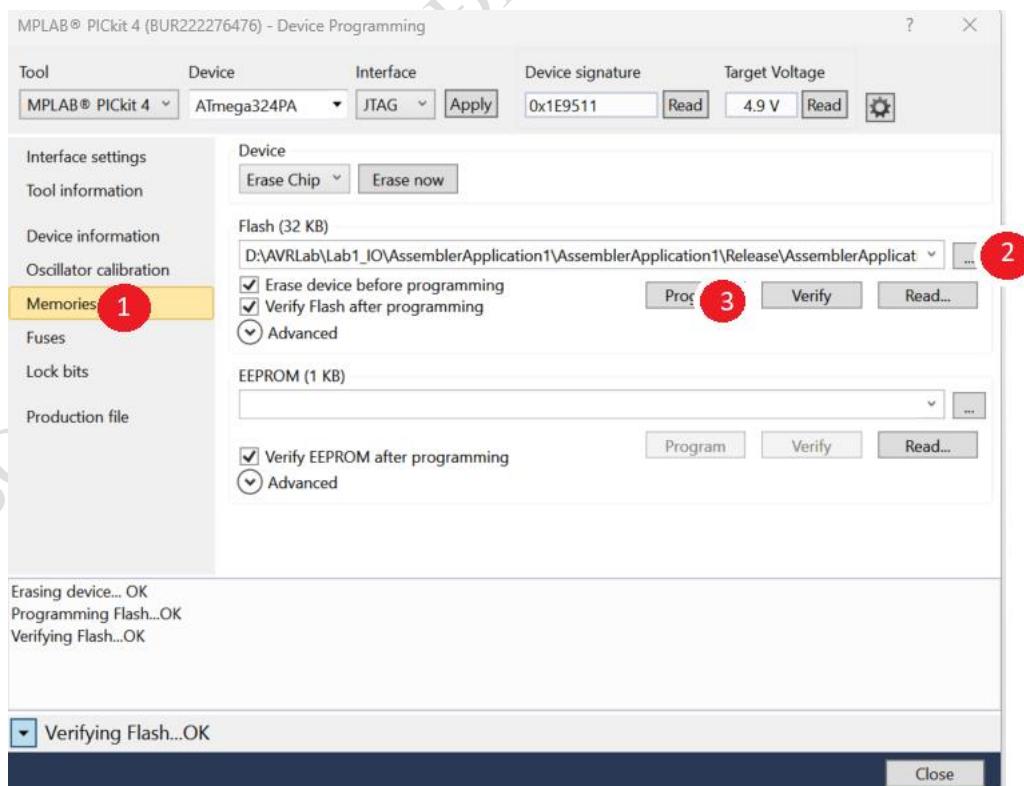
Trong mục Tool, chọn MPLAB PICKIT 4, chọn giao diện JTAG hoặc ISP trong mục Interface (tùy theo kết nối hiện tại). Kích chọn Apply. Tại mục Device Signature, kích chọn Read. Tại Target Voltage, chọn Read.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ



7. Lập trình cho chip

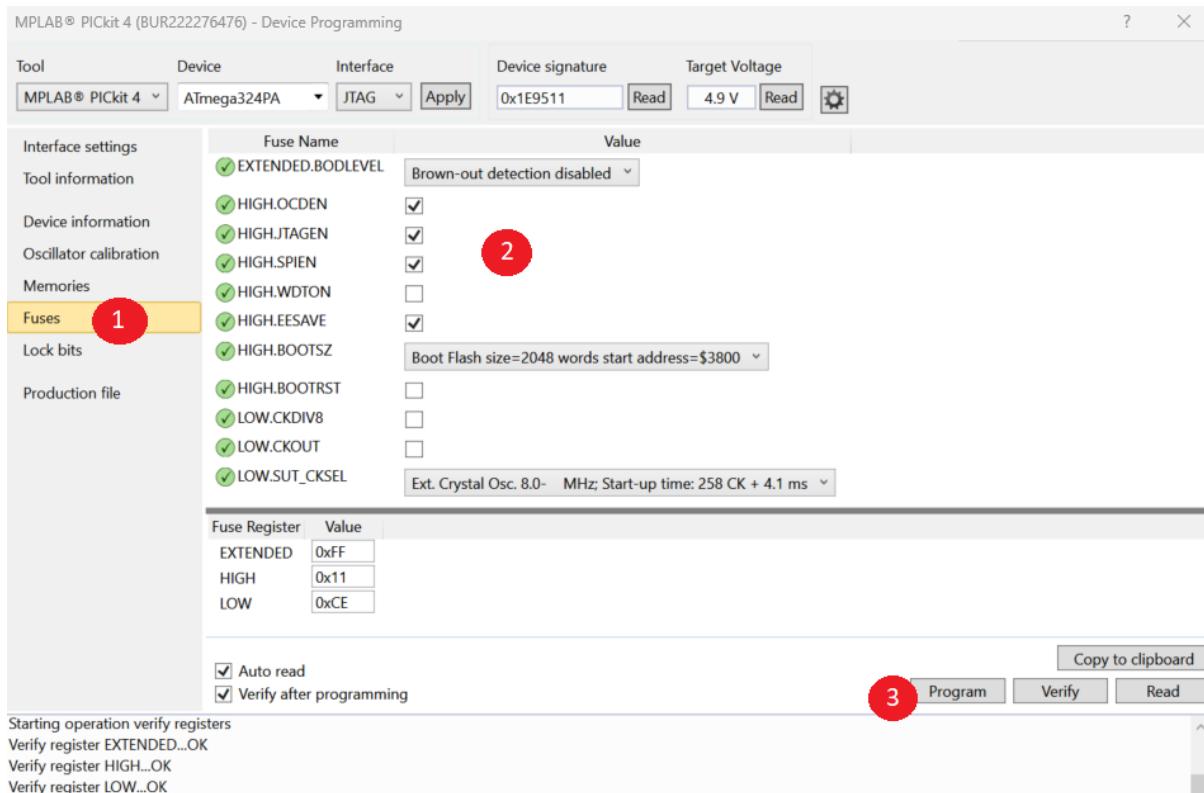
Chọn Memories. Trong phần Flash, chọn file Hex đã được biên dịch. Kích chọn Program để nạp chương trình.



HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

8. Lập trình cho cầu chì

Nếu muốn thay đổi các cấu hình hoạt động của chip, ta cần lập trình cho cầu chì. Kích chọn Fuses, chọn các cầu chì muốn lập trình và chọn Program



9. Ý nghĩa các bit cầu chì

Khi gán **giá trị 0** cho 1 Fuse bit đồng nghĩa với Fuse bit đó được lập trình (**programmed**), trong khi **1** nghĩa là không được lập trình (**unprogrammed**).

High Fuse Byte	Bit No.	Description	Default Value
OCDEN ⁽¹⁾	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN ⁽²⁾	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
WDTON ⁽³⁾	4	Watchdog Timer Always On	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed), EEPROM not reserved
BOOTSZ1 ⁽⁴⁾	2	Select Boot Size	0 (programmed)
BOOTSZ0 ⁽⁴⁾	1	Select Boot Size	0 (programmed)
BOOTRST	0	Boot Reset vector Enabled	1 (unprogrammed)

- Bit 7 (OCDEN) :** kích hoạt chức năng OCD (onchip debugger). Mặc định OCD bị disable. Để thực hiện chức năng debug dùng JTAG thì bit này phải được lập trình.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

- Bit 6 (JTAGEN)** : khi bit này bằng 0 đồng nghĩa với cho phép sử dụng các chân jtag để debug hoặc nạp chương trình,khi đó các chân jtag này không được sử dụng như các I/O port bình thường được nữa. Mặc định JTAG được enable.
Để dùng JTAG debug chương trình thì OCDEN phải được lập trình.
- Bit 5 (SPIEN)** : Cho phép lập trình ISP sử dụng SPI.
- Bit 4 (WDTON)** : Enable watchdog timer
- Bit 3 (EESAVE)** : cho phép để lại (lưu lại) dữ liệu EEPROM trong mỗi lần nạp lại chip hay không, nếu **EESAVE=0** thì toàn bộ dữ liệu trong EEPROM sẽ không bị xóa khi nạp chip hoặc ngược lại.
- Bit 2 (BOOTSZ1) , Bit 1 (BOOTSZ0):** là 2 bit chọn phân vùng bộ nhớ:

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	256 words	4	0x0000 - 0x3EFF	0x3F00 - 0x3FFF	0x3EFF	0x3F00
1	0	512 words	8	0x0000 - 0x3DFF	0x3E00 - 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 - 0x3BFF	0x3C00 - 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 - 0x37FF	0x3800 - 0x3FFF	0x37FF	0x3800

- Bit 0 (BOOTRST)** : cho phép chạy chương trình bootloader trong vùng lưu trữ bootloader. **BOOTRST=1** thì địa chỉ của chương trình sẽ bắt đầu từ địa chỉ 0x0000, còn **BOOTRST=0** thì ô nhớ bắt đầu là vị trí đầu tiên của bootloader tức chương trình bootloader được thực thi.

Low Fuse Byte	Bit No.	Description	Default Value
CKDIV8 ⁽⁴⁾	7	Divide clock by 8	0 (programmed)
CKOUT ⁽³⁾	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed) ⁽¹⁾
SUT0	4	Select start-up time	0 (programmed) ⁽¹⁾
CKSEL3	3	Select Clock source	0 (programmed) ⁽²⁾
CKSEL2	2	Select Clock source	0 (programmed) ⁽²⁾
CKSEL1	1	Select Clock source	1 (unprogrammed) ⁽²⁾
CKSEL0	0	Select Clock source	0 (programmed) ⁽²⁾

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

- Bit 7 (CKDIV8) : Chia tần số dao động cho 8.
- Bit 3 (CKSEL[3,2,1,0]) : Chọn nguồn xung nhịp :

Device Clocking Option	CKSEL[3:0]
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Bảng 1 Cấu hình cấu chì cho AVR

Khi CKSEL[3:0] = 0010 khi đó nguồn dao động RC on chip được chọn và tần số dao động từ 7,3 – 8MHZ tùy theo nhiệt độ chip , nếu bit CKDIV = 0 , thì tần số dao động được chọn là 8MHZ/8 = 1MHZ , ở mặc định thì Vi xử lý chọn dao động nội giá trị 1MHZ.

Có thể chọn 1 trong các giá trị này CKSEL[3:0] = 0111-0110 để chọn nguồn dao động thạch anh ngoài có giá trị lên đến 20MHz.

Extended Fuse Byte	Bit No.	Description	Default Value
-	5	-	0
-	4	-	0
-	3	-	0 (0: Disabled)
BODLEVEL2 ⁽¹⁾	2	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL1 ⁽¹⁾	1	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0 ⁽¹⁾	0	Brown-out Detector trigger level	1 (unprogrammed)

Khi nguồn nuôi Vi xử lý sụt xuống mức điện áp nhất định thì vi xử lý sẽ tự reset

BODLEVEL [2:0] Fuses	Min. V _{BOT}	Typ V _{BOT}	Max V _{BOT}	Units
111	BOD Disabled			
110	1.7	1.8	2.0	V
101	2.5	2.7	2.9	
100	4.1	4.3	4.5	
011	Reserved			
010				
001				
000				

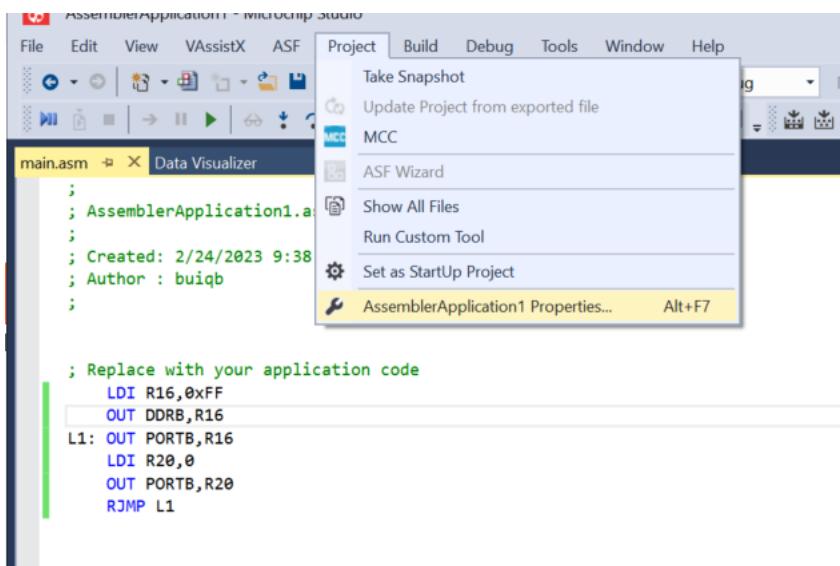
HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Lưu ý: Kit thí nghiệm đã được lập trình cầu chì sẵn cho phép debug chương trình sử dụng JTAG port, với dao động lấy từ thạch anh 8Mhz bên ngoài, không chia cho 8. Vì vậy xung nhịp cho CPU sẽ là 8Mhz, tương ứng $1MC=0.125\mu s$.

10. Gỡ rối chương trình

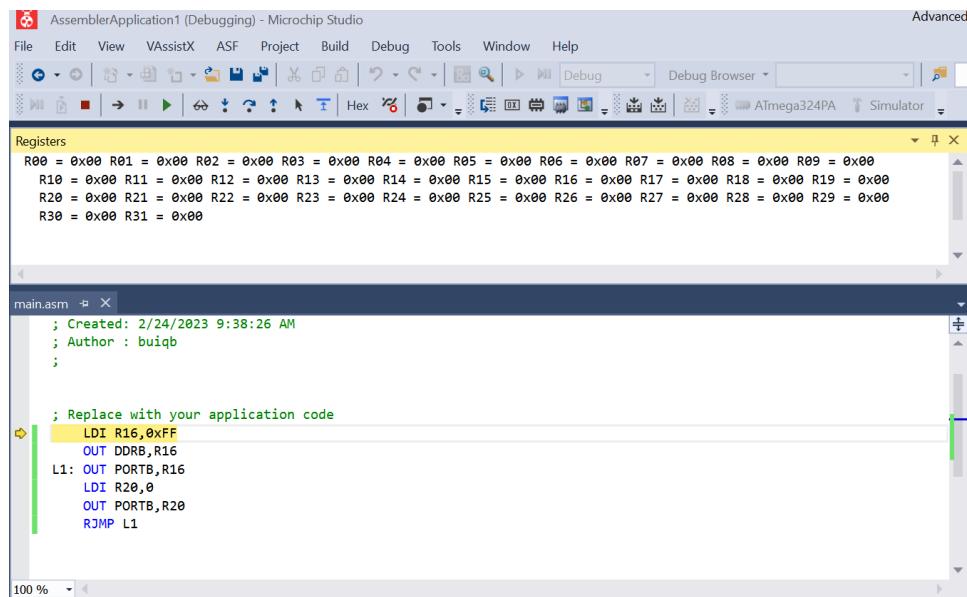
Ta có thể debug chương trình bằng cách mô phỏng (simulate) hoặc chạy trực tiếp trên vi điều khiển, sử dụng bộ gỡ rối PICKIT 4. Khi đó, phải kết nối PICKIT4 vào cổng JTAG.

a. Chọn Project-Property

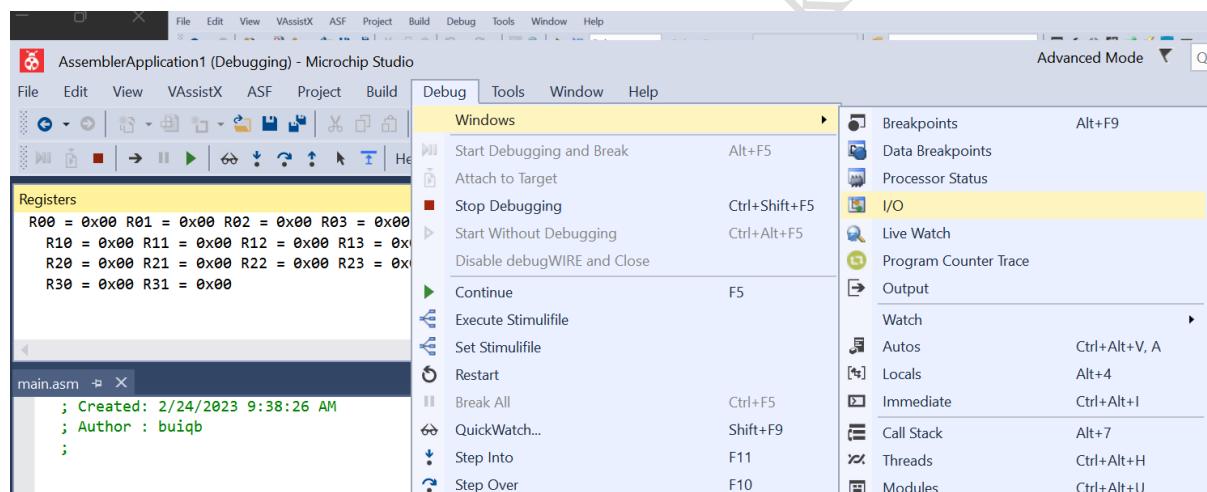


- b. Chọn Tool. Tại mục Select Debugger/Programmer, chọn Simulator nếu muốn mô phỏng hoặc chọn MPLAB PICKIT4 nếu muốn gỡ rối trực tiếp trên chip. Nhấn Ctrl-S để lưu cấu hình.
- c. Chọn Debug-Start Debugging and Break. Chương trình dừng ở dòng lệnh đầu tiên với một line màu vàng và sẵn sàng để debug.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

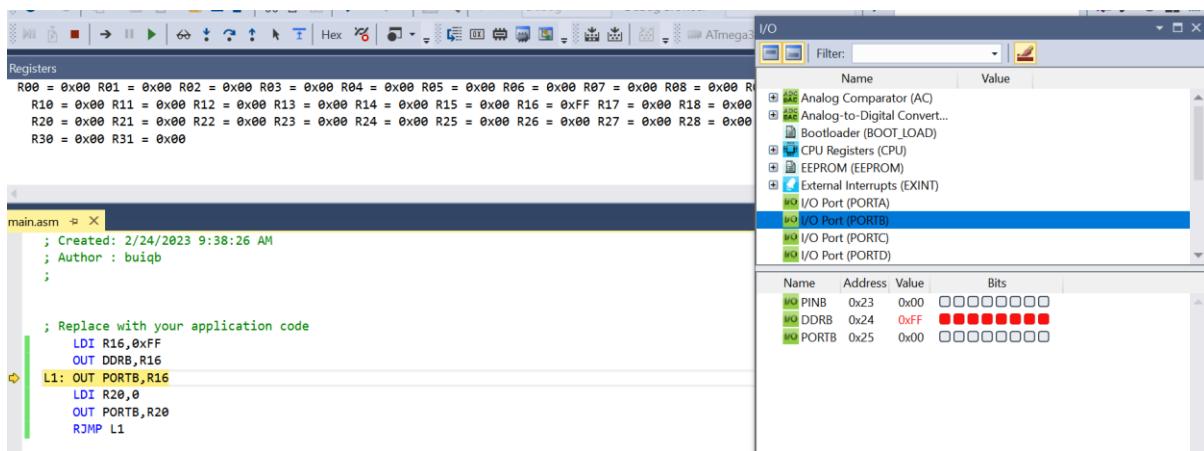


- d. Để hiển thị cửa sổ trạng thái các ngoại vi, chọn Debug-Windows và chọn các cửa sổ mong muốn.



- e. Chọn cửa sổ I/O và chọn PORT B để quan sát giá trị các thanh ghi liên quan đến PORT B. Nhấn F10 hoặc chọn Debug/Step Over để xem quá trình thực hiện chương trình và sự thay đổi giá trị của các thanh ghi PORTB

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ



f. Để chạy chương trình từng bước ta nhấn F10 hoặc nhấn vào icon Step Over

Step into vs Step Over

Cả 2 chức năng Step Over (F10) và Step Into (F11) đều chạy 1 lệnh và đi đến lệnh kế tiếp. Tuy nhiên, khi lệnh đang thực thi là một lệnh gọi chương trình con, Step Into sẽ đi đến lệnh đầu tiên của chương trình con, còn Step Over sẽ thực hiện toàn bộ chương trình con và đi đến lệnh kế tiếp lệnh Call.

Step Out

Nếu đang thực hiện một chương trình con, ta có thể thực hiện hết chương trình con bằng lệnh Step Out

Run to cursor

Ta có thể click phải chuột vào trước một lệnh, sau đó chọn Run To Cursor. Chương trình sẽ chạy cho đến khi nào tới được lệnh đang được chỉ bởi con trỏ chuột

Break Point

Ta có thể cho chương trình chạy và dừng khi gặp một vị trí xác định trước bằng cách đặt Break Point tại vị trí muốn dừng.

Đặt Break Point bằng cách click chuột phải vào vị trí mong muốn và chọn Break Point-Insert BreakPoint, hoặc bằng cách click chuột trái tại vị trí đầu dòng lệnh trên thanh đọc màu xám để chọn/bỏ chọn breakpoint.

Chi tiết cách sử dụng Simulator và Debugger, sinh viên tham khảo tài liệu: Getting-Started-with-Microchip-Studio-DS50002712B.pdf, mục 1.12: AVR MCU Simulator Debugging.

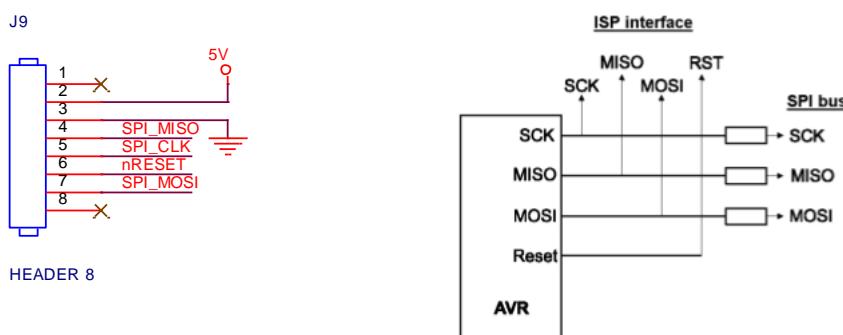
1.5 Sử dụng các chân port lập trình ISP và JTAG

1.5.1 Giao diện lập trình ISP:

Header J9 cho phép kết nối bộ PICKIT vào để lập trình cho MCU qua giao diện SPI. Để lập trình qua giao diện này thì bit cầu chì SPIEN phải được lập trình.

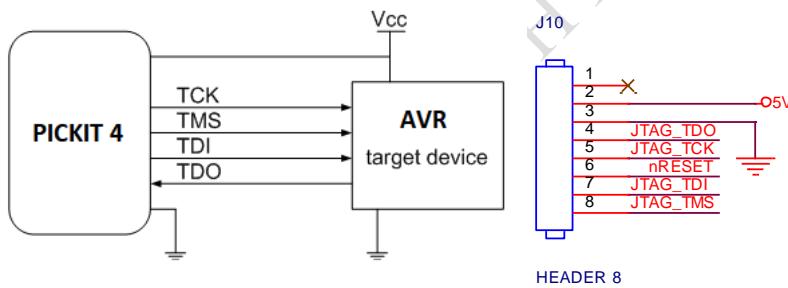
Sau khi download chương trình xuống kit, các tín hiệu SPI trên AVR được sử dụng bình thường.

Lưu ý rằng ta không thể gỡ rối chương trình qua giao diện ISP.



Hình 4 Sử dụng các chân SPI để lập trình

1.5.2 Giao diện gỡ rối JTAG:



Hình 5 Giao diện JTAG

Để có thể gỡ rối chương trình, ta phải sử dụng giao diện JTAG, với các tín hiệu như trên Hình 5. Trên kit thí nghiệm, header J10 được sử dụng để kết nối bộ PICKIT 4.

MCU	JTAG Signal
PC5	TDI
PC4	TDO
PC3	TMS
PC2	TCK

Có thể thấy khi giao diện JTAG được cho phép, các chân PC5..PC2 không thể được sử dụng. Để sử dụng các chân này, ta phải disable bit cầu chì **JTAGEN**. Khi đó, ta chỉ có thể sử dụng giao diện lập trình ISP thông qua các chân SPI.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

BỘ MÔN ĐIỆN TỬ - ĐH BÁCH KHOA TP.HCM

CHƯƠNG 2 NÚT NHẤN VÀ LED ĐƠN

2.1 LÝ THUYẾT CƠ BẢN

Trong thực tế, led đơn và nút nhấn là hai phương pháp giao tiếp người sử dụng đơn giản và cũng rất hiệu quả. Người sử dụng có thể dùng nút nhấn tác động vào hệ thống điều khiển để ra lệnh cho hệ thống. Ngược lại, led đơn có thể được dùng để hiển thị trạng thái hoạt động bên trong của hệ thống. Led đơn đặc biệt hữu dụng trong việc giúp người thiết kế gỡ rối chương trình. Ví dụ người lập trình có thể thay đổi tốc độ nhấp nháy của led để chỉ ra các lỗi khác nhau trong chương trình.

2.2 THIẾT KẾ PHẦN CỨNG

2.2.1 Khối nút nhấn đơn

Công tắc (Switch) là linh kiện cơ khí được dùng để nối 2 hoặc nhiều tiếp điểm lại với nhau khi có tác động. Thông thường Switch có 2 tiếp điểm (Switch đơn).

Để giao tiếp switch đơn có hai cách giao tiếp như sau:

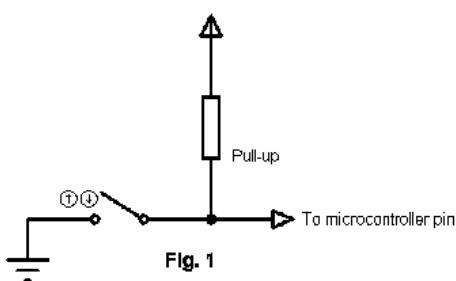


Fig. 1

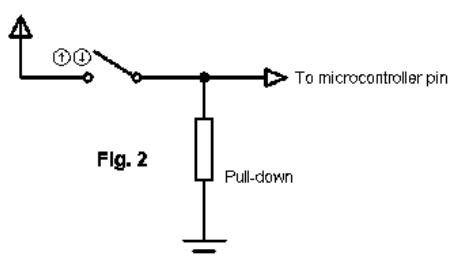


Fig. 2

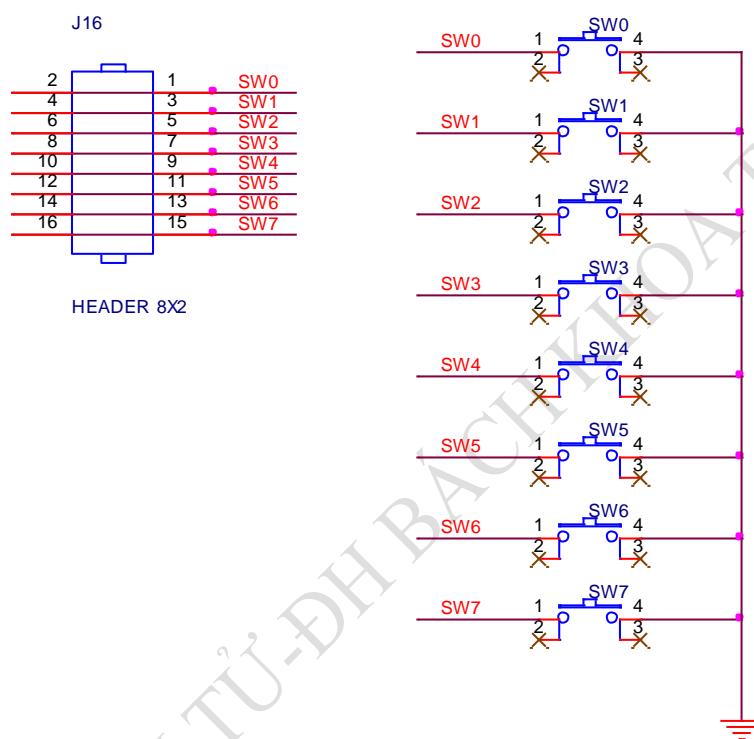
Hình 6 Cách giao tiếp switch đơn

Ở cách phía trên, một điện trở kết nối nguồn VCC và 1 đầu của switch, đầu này lấy tín hiệu ra đưa vào chân vi điều khiển, đầu còn lại nối với GND. Khi Switch không đóng, tín hiệu ra sẽ là

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

HIGH (logic 1). Khi Switch đóng, tín hiệu ra sẽ là LOW (logic 0). Cách kết nối này là sử dụng điện trở kéo lên (pull-up resistor).

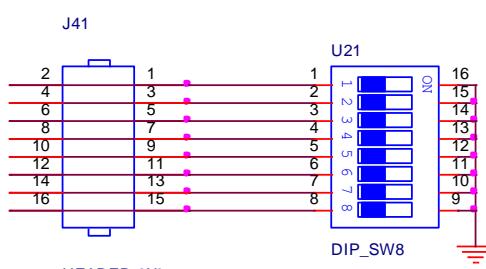
Cách giao tiếp còn lại thì hoàn toàn ngược lại, dùng điện trở kéo xuống (pull-down resistor). Trên kit thí nghiệm có 8 nút nhấn đơn, được kết nối vào header J16. Để tương tác với nút nhấn, ta phải kết nối các tín hiệu từ J16 đến chân port tương ứng, sử dụng dây đơn hoặc bus dây để kết nối nguyên PORT. Lưu ý, trên thiết kế chưa có điện trở kéo lên bên ngoài.



Hình 7 Sơ đồ kết nối nút nhấn đơn

2.2.2 Khối Dip Switch

Trên kit có 1 DIP Switch gồm 8 switch, được kết nối vào header J41. Để tương tác với nút nhấn, ta phải kết nối các tín hiệu từ J41 đến chân port tương ứng, sử dụng dây đơn hoặc bus dây để kết nối nguyên PORT. Lưu ý, trên thiết kế chưa có điện trở kéo lên bên ngoài.

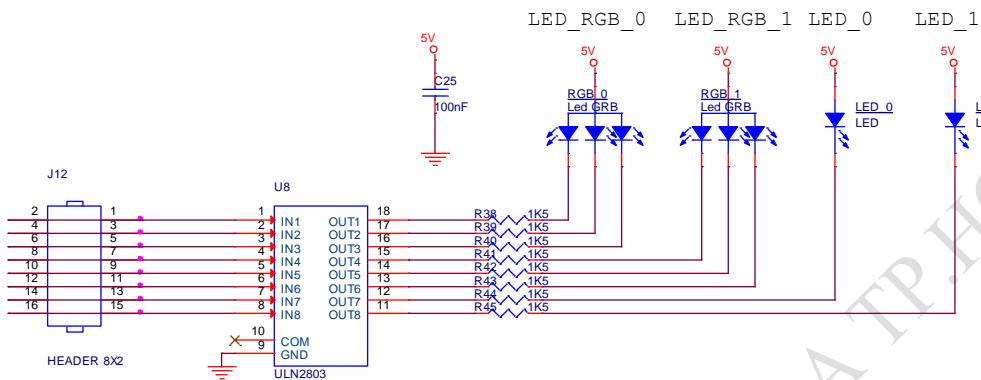


Hình 8 Sơ đồ kết nối DIP Switch

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

2.2.3 Khối LED đơn

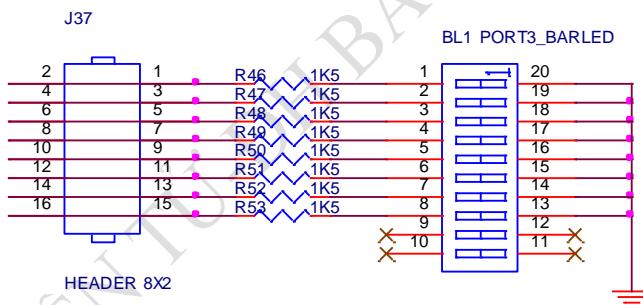
Kit thí nghiệm hỗ trợ 2 LED RGB và 2 LED đơn, được kết nối vào J12 và được lái bởi IC ULN2803. Để bật sáng 1 LED, tín hiệu tương ứng trên J12 sẽ bằng 1.



Hình 9 Sơ đồ kết nối khối LED đơn

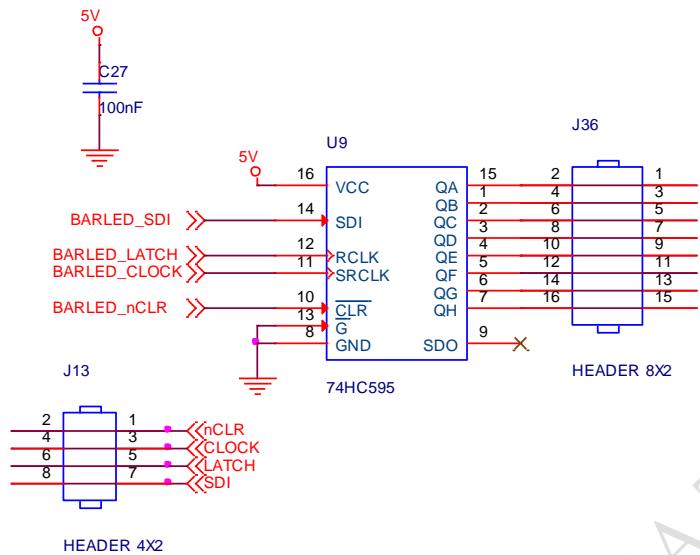
2.2.4 Khối BAR LED

1 bar LED gồm 8 LED đơn được kết nối vào J37. Để LED sáng, tín hiệu tương ứng trên J37 bằng 1.



Hình 10 Sơ đồ kết nối khối bar LED

2.2.5 Khối thanh ghi dịch



Hình 11 Sơ đồ khối thanh ghi dịch

Để có thể mở rộng số lượng tín hiệu output, trên kit thí nghiệm hỗ trợ IC thanh ghi dịch 74HC595 với sơ đồ kết nối như Hình 11.

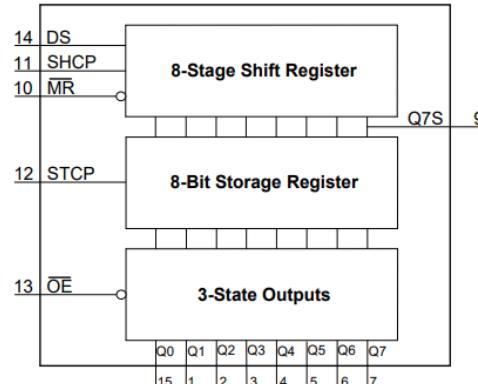
Để sử dụng khối thanh ghi dịch, các tín hiệu điều khiển trên J13 được nối vào chân PORT của vi điều khiển, các ngõ ra trên J36 được nối vào các LED đơn hoặc bar led.

Dữ liệu được dịch ra tuần tự, sau đó được chốt và xuất ra theo giản đồ xung như Hình 12

Pin Descriptions

Pin Number	Pin Name	Function
1	Q1	Parallel Data Output 1
2	Q2	Parallel Data Output 2
3	Q3	Parallel Data Output 3
4	Q4	Parallel Data Output 4
5	Q5	Parallel Data Output 5
6	Q6	Parallel Data Output 6
7	Q7	Parallel Data Output 7
8	GND	Ground
9	Q7S	Serial Data Output
10	MR	Master Reset Input
11	SHCP	Shift Register Clock Input
12	STCP	Storage Register Clock Input
13	OE	Output Enable Input
14	DS	Serial Data Input
15	Q0	Parallel Data Output 0
16	V _{CC}	Supply Voltage

Functional Diagram



HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Control				Input	Output		Function
SHCP	STCP	OE	MR	DS	Q7S	Qn	
X	X	L	L	-	L	NC	Low-level asserted on MR clears shift register. Storage register is unchanged.
X	↑	L	L	-	L	L	Empty shift register transferred to storage register.
X	X	H	L	-	L	Z	Shift register remains clear; All Q outputs in Z state.
↑	X	L	H	-	Q6S	NC	HIGH is shifted into first stage of Shift Register Contents of each register shifted to next register. The content of Q6S has been shifted to Q7S and now appears on device pin Q7S.
X	↑	L	H	-	NC	QnS	Contents of shift register copied to storage register. With output now in active state the storage register contents appear on Q outputs.
↑	↑	L	H	-	Q6S	QnS	Contents of shift register copied to output register then shift register shifted.

H=HIGH Voltage State

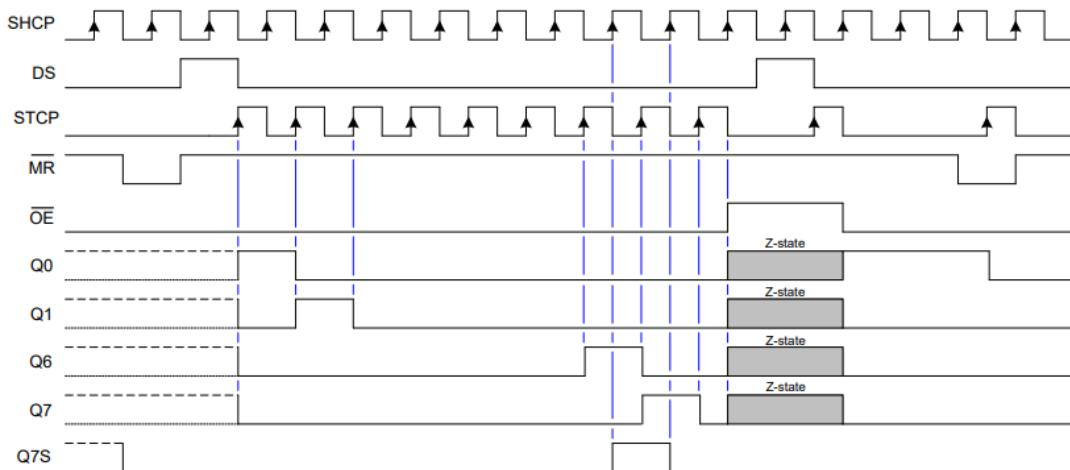
L=LOW Voltage State

↑=LOW to HIGH Transition

X= Don't Care – High or Low (Not Floating)

NC= No Change

Z= High-Impedance State



Hình 12 Mô tả hoạt động của thanh ghi dịch 74HC595

2.3 LẬP TRÌNH GIAO TIẾP LED VÀ SWITCH

Để sử dụng các khối switch và Led hiển thị, sinh viên sử dụng dây bus hoặc dây đơn kết nối các ngoại vi đến các chân port của AVR. Sau đây là một chương trình ví dụ đọc trạng thái switch gắn vào chân PB.0 và hiển thị ra LED gắn vào chân PC.0.

```
.cseg
.org 0x00
start:
        CBI      DDRB,0      ;PB0 is input
        SBI      PORTB,0     ;enable pull up resistor
        SBI      DDRC,0      ;PC0 is output
        CBI      PORTC,0     ;Turn off the LED

MAIN:
        SBIS    PINB,0      ;if swswitch not pressed, skip the jump to release
        RJMP    RELEASE

PRESSED:
        CBI      PORTC,0     ;turn on the LED
        RJMP    MAIN

RELEASE:
        SBI      PORTC,0     ;turn off the LED
        RJMP    MAIN
```

Ví dụ 1 Giao tiếp nút nhấn đơn và LED đơn dùng assembly

Tương tự, ta có thể sử dụng C để viết chương trình với cùng chức năng:

```
#include <avr/io.h>
int main(void)
{
    DDRC = 0X01;
    PORTB |= 0X01;
    /* Replace with your application code */
    while (1)
    {
        if (PINB & (1<<PINB0))      PORTC &= ~(1<<PORTC0);
        else PORTC |= (1<<PORTC0);
    }
}
```

Ví dụ 2 Giao tiếp nút nhấn và LED dùng C

2.4 LẬP TRÌNH TẠO ĐỘ TRỄ DÙNG CÂU LỆNH

Trong thực tế, để điều khiển các ngoại vi, các thao tác điều khiển luôn cần phải tuân theo một trình tự nhất định với khoảng thời gian giữa chúng là xác định. Do đó, các chương trình con tạo trễ luôn là một phần rất quan trọng của chương trình điều khiển.

Ví dụ như ta cần bật/tắt 1 LED để báo hiệu là chương trình đang hoạt động. Nếu LED bật/tắt quá nhanh, ta sẽ có cảm giác là LED luôn sáng. Vì vậy, giữa 2 lần sáng/tắt cần có 1 thời gian trễ để mắt người có thể cảm nhận được, thường thời gian này là 1 s.

Có nhiều cách để có thể viết được chương trình con tạo trễ. Phần thí nghiệm này giúp người lập trình nắm được cách thức tạo trễ dùng các câu lệnh tạo thành vòng lặp. Mỗi vi điều khiển luôn sử dụng một tín hiệu xung clock để đồng bộ các hoạt động trong hệ thống. Một câu lệnh được thực thi sẽ cần một số xung clock xác định thường được đo bằng chu kỳ máy. Như vậy, một câu lệnh được thực thi sẽ tiêu hao một khoảng thời gian xác định.

Vì điều khiển Atmega 324 có thể được cấp nhiều nguồn xung nhịp khác nhau, tùy theo cách cấu hình cầu chì như ở Bảng 1. Ví dụ sau mô tả cách viết chương trình con `DELAY_10ms` với tần số clock là 8MHZ.

```
.cseg
.org 0x00
START:
    SBI      DDRC, 0
MAIN:
    CBI      PORTC, 0
    CALL    DELAY_10MS
    SBI      PORTC, 0
    CALL    DELAY_10MS
    JMP     MAIN
DELAY_10MS:
    LDI     R21, 80      ;1MC
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

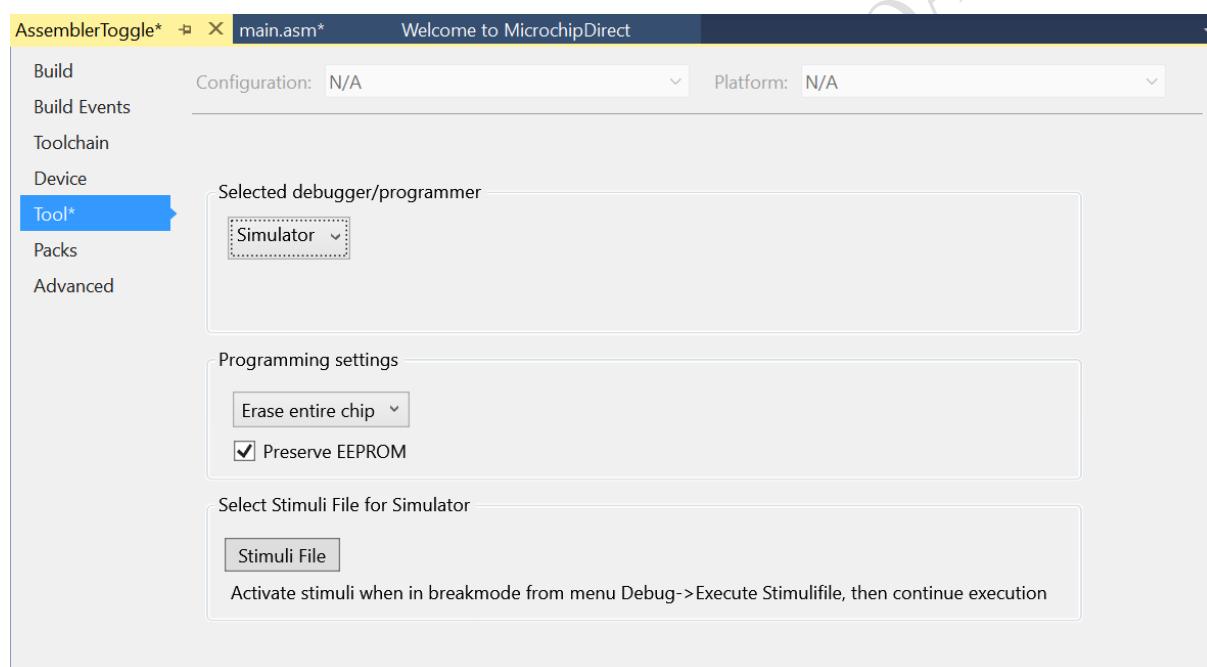
```
L1:      LDI      R20,250    ;1MC
L2:      DEC      R20       ;1MC
        NOP      ;1MC
        BRNE    L2       ;2/1MC
        DEC      R21       ;1MC
        BRNE    L1       ;2/1MC
        RET      ;4MC
```

Ví dụ 3: tạo trễ dùng vòng lặp (assembly)

Trên đây là ví dụ tạo một xung có tần số 50Hz trên chân PC0. Chương trình con **DELAY_10MS** tạo ra một thời gian trễ 10ms với clock **8MHZ, 1MC=0.125μs**, ứng với $80000\text{MC} \times 0.125 = 10\text{ms}$.

Để biết chính xác một đoạn chương trình chạy trong bao nhiêu clock, ta sử dụng tính năng Cycle Counter trong AVR Simulator.

1. Chọn chức năng Simulator



2. Chọn Debug-Start Debugging and Break để bắt đầu mô phỏng. Chọn Debug-Windows-Processor Status để mở cửa sổ Processor Status. Đặt BreakPoint ở bắt đầu và kết thúc của khối lệnh mà ta muốn đo, ở đây là chương trình con **DELAY_10MS**

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Registers

```
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00 R07 = 0x00
R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x00 R15 = 0x00 R16 = 0x00 R17 = 0x00
R20 = 0x00 R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00
R30 = 0x00 R31 = 0x00
```

Processor Status

Name	Value
Program Counter	0x00000000
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	1.000 MHz
Stop Watch	0.00 µs

AssemblerToggle main.asm Welcome to MicrochipDirect

```
.cseg
.org 0x00
START:
    SBI DDRC, 0
MAIN:
    CBI PORTC, 0
    CALL DELAY_10MS
    SBI PORTC, 0
    CALL DELAY_10MS
    JMP MAIN
DELAY_10MS:
    LDI R21, 10 ;1MC
L1:   LDI R20, 249 ;1MC
L2:   DEC R20 ;1MC
```

Registers

R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00

3. Cho chương trình chạy đến Break Point thứ nhất, click vào Cycle Counter và reset giá trị này về 0.

Registers

```
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00 R07 = 0x00
R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x00 R15 = 0x00 R16 = 0x00 R17 = 0x00
R20 = 0x00 R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00
R30 = 0x00 R31 = 0x00
```

Processor Status

Name	Value
Program Counter	0x00000002
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	3
Frequency	1.000 MHz
Stop Watch	3.00 µs

AssemblerToggle main.asm Welcome to MicrochipDirect

```
.cseg
.org 0x00
START:
    SBI DDRC, 0
MAIN:
    CBI PORTC, 0
    CALL DELAY_10MS
    SBI PORTC, 0
    CALL DELAY_10MS
    JMP MAIN
DELAY_10MS:
    INT DD1 10 ;1MC
```

Registers

R00	0x00
R01	0x00
R02	0x00
R03	0x00

4. Tiếp tục cho chương trình chạy đến Break Point thứ 2 và ghi nhận lại giá trị Cycle Count. Đó là số clock cần thiết để chạy đoạn chương trình này.

Registers

```
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00 R07 = 0x00
R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x00 R15 = 0x00 R16 = 0x00 R17 = 0x00
R20 = 0x00 R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00
R30 = 0x00 R31 = 0x00
```

Processor Status

Name	Value
Program Counter	0x00000004
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	10001
Frequency	1.000 MHz
Stop Watch	10,001.00 µs

AssemblerToggle main.asm Welcome to MicrochipDirect

```
.cseg
.org 0x00
START:
    SBI DDRC, 0
MAIN:
    CBI PORTC, 0
    CALL DELAY_10MS
    SBI PORTC, 0
    CALL DELAY_10MS
    JMP MAIN
DELAY_10MS:
    LDI R21, 10 ;1MC
L1:   LDI R20, 249 ;1MC
L2:   DEC R20 ;1MC
```

Registers

R00	0x00
R01	0x00
R02	0x00

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Nếu ta sử dụng C để lập trình, ta có thể sử dụng các vòng lặp đơn giản như sau:

```
int main(void)
{
    volatile int i;
    /* Replace with your application code */
    DDRC |= 0x01;
    while (1)
    {
        PORTC ^= 0x01;
        for (i=554;i>0;i--);
    }
}
```

Ví dụ 4: tạo trễ dùng vòng lặp (C)

Số đếm nạp vào biến đếm để có thời gian trễ mong muốn được tinh chỉnh sử dụng Simulator như đã nói ở phần trên.

Ngoài ra, ta có thể sử dụng thư viện có sẵn. Lưu ý, để dùng được các hàm trong thư viện delay.h ta phải định nghĩa tần số CPU cho đúng. Trong ví dụ dưới đây, tần số CPU là 1 Mhz

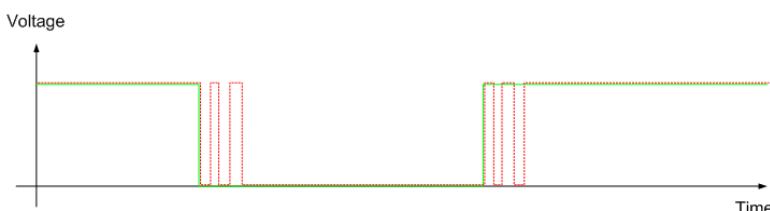
```
#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>

int main(void)
{
    volatile int i;
    /* Replace with your application code */
    DDRC |= 0x01;
    while (1)
    {
        PORTC ^= 0x01;
        _delay_ms(10);
    }
}
```

Ví dụ 5: tạo trễ dùng thư viện delay

2.5 RUNG PHÍM VÀ CHỐNG RUNG

Với phím nhấn cơ khí, tiếp điểm cơ khí sau khi được nhấn hay nhả sẽ bị rung động. Do đó, khi nhấn hay nhả phím sẽ tạo ra một chuỗi các xung thay vì một xung đơn như ở Hình 13



Hình 13 Phím đơn bị rung khi nhấn hay nhả

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Khi đó, nếu ta dùng tín hiệu này để xác định số lần phím nhấn thì kết quả sẽ bị sai lệch. Để có kết quả đúng ta cần chống rung (debounce).

Cách chống rung đơn giản nhất là đọc tín hiệu ra hai lần liên tiếp sau một khoảng thời gian trễ thích hợp, nếu 2 lần này mà kết quả đọc được giống nhau thì ta cho là kết quả đúng, nếu kết quả là sai thì ta lại đọc lần thứ 3 cũng sau thời gian T, nếu kết quả giống như lần đọc thứ 2 thì coi như kết quả lần này là đúng, còn nếu không thì lại lặp lại quá trình. Thời gian T thường được chọn là 50ms.

Ta có thể tăng độ chính xác bằng cách giảm thời gian T và tăng số lần đọc lên. Ví dụ phím được coi là trả về giá trị 1 nếu như ta đọc được liên tiếp 20 lần cùng giá trị 1 với mỗi lần đọc cách nhau 1ms.

2.6 LẬP TRÌNH GIAO TIẾP KHỐI THANH GHI DỊCH

Khối thanh ghi dịch được sử dụng trong trường hợp ta muốn mở rộng các tín hiệu hiển thị mà không cần phải sử dụng nhiều chân port. Các thanh ghi dịch này có thể được mắc nối tiếp với nhau để tăng số lượng tín hiệu.

Dữ liệu được dịch nối tiếp vào thanh ghi, và được xuất ra các chân ngõ ra của IC74HC595.

Đang sóng điều khiển được mô tả ở phần 2.2.5

Ví dụ về điều khiển thanh ghi dịch để xuất dữ liệu, sử dụng assembly và C. Các chân kết nối với vi điều khiển AVR như bảng sau. Kết nối header ngõ ra của thanh ghi dịch với header của BARLED.

PB0	SDI
PB1	LATCH
PB2	CLOCK
PB3	MCLR

```
.include "m324padef.inc" ; Include Atmega324pa definitions
.def shiftData = r20 ; Define the shift data register
.equ clearSignalPort = PORTB ; Set clear signal port to PORTB
.equ clearSignalPin = 3 ; Set clear signal pin to pin 0 of PORTB
.equ shiftClockPort = PORTB ; Set shift clock port to PORTB
.equ shiftClockPin = 2 ; Set shift clock pin to pin 1 of PORTB
.equ latchPort = PORTB ; Set latch port to PORTB
.equ latchPin = 1 ; Set latch pin to pin 0 of PORTB
.equ shiftDataPort = PORTB ; Set shift data port to PORTB
.equ shiftDataPin = 0 ; Set shift data pin to pin 3 of PORTB
main:
    call initport
    ldi shiftData, 0x55
    call cleardata
    call shiftoutdata
stop:
    jmp stop
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
; Initialize ports as outputs
initport:
    ldi r24,
    (1<<clearSignalPin)|(1<<shiftClockPin)|(1<<latchPin)|(1<<shiftDataPin)
        out DDRB, r24          ; Set DDRB to output
        ret
        ldi    shiftData,0x55
cleardata:
    cbi clearSignalPort, clearSignalPin      ; Set clear signal pin to low
; Wait for a short time
    sbi clearSignalPort, clearSignalPin      ; Set clear signal pin to high
    ret
; Shift out data
shiftoutdata:
    cbi shiftClockPort, shiftClockPin        ;
    ldi r18, 8                      ; Shift 8 bits
shiftloop:
    sbrc shiftData, 7      ; Check if the MSB of shiftData is 1
    sbi shiftDataPort, shiftDataPin    ; Set shift data pin to high
    sbi shiftClockPort, shiftClockPin ; Set shift clock pin to high
    lsl shiftData           ; Shift left
    cbi shiftClockPort, shiftClockPin ; Set shift clock pin to low
    cbi shiftDataPort, shiftDataPin  ; Set shift data pin to low
    dec r18
    brne shiftloop
; Latch data
    sbi latchPort, latchPin   ; Set latch pin to high
    cbi latchPort, latchPin   ; Set latch pin to low
    ret
```

Ví dụ 6: Mã assembly để dịch dữ liệu ra 74HC595

```
#include <avr/io.h>
#define      F_CPU  1000000UL
#include <util/delay.h>

#define clearSignalPort PORTB
#define clearSignalPin 3

#define shiftClockPort PORTB
#define shiftClockPin 2

#define latchPort PORTB
#define latchPin 1
#define shiftDataPort PORTB
#define shiftDataPin 0

void initializePorts() {
    // Set output pins
    DDRB |= (1 << clearSignalPin)|(1 << shiftClockPin) | (1 << latchPin)|(1 <<
shiftDataPin);
}

void clearShiftRegister() {
    // Set clear signal pin to low
    clearSignalPort &= ~(1 << clearSignalPin);
    _delay_us(1);
    // Set clear signal pin to high
    clearSignalPort |= (1 << clearSignalPin);
}
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
void shiftOut(uint8_t data) {
    // Set latch pin to low
    latchPort &= ~(1 << latchPin);
    // Shift out 8 bits
    for (int i = 0; i < 8; i++) {
        if (data & 0x80) {
            // Set shift data pin to high
            shiftDataPort |= (1 << shiftDataPin);
        } else {
            // Set shift data pin to low
            shiftDataPort &= ~(1 << shiftDataPin);
        }
        // Set shift clock pin to high
        shiftClockPort |= (1 << shiftClockPin);
        // Set shift clock pin to low
        shiftClockPort &= ~(1 << shiftClockPin);
        // Shift data left by 1 bit
        data <<= 1;
    }

    // Set latch pin to high
    latchPort |= (1 << latchPin);
    _delay_us(1);
    // Set latch pin to low
    latchPort &= ~(1 << latchPin);}
int main() {
    uint8_t data = 0x55;
    initializePorts();
    clearShiftRegister();
    shiftOut(data);
    while (1) {
        // Your main program loop goes here
        _delay_ms(1000);
        data = (data << 1) | (data >> 7);
        shiftOut(data);
    }
    return 0;}
```

Ví dụ 7: Mã C để dịch dữ liệu ra 74HC595

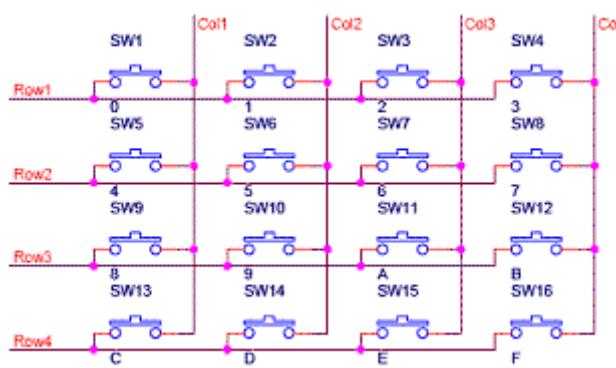
BỘ MÔN ĐIỆN TỬ & HỆ THỐNG TIN HỌC
BỘ MÔN ĐIỆN TỬ & HỆ THỐNG TIN HỌC

CHƯƠNG 3 BÀN PHÍM MA TRẬN

3.1 LÝ THUYẾT CƠ BẢN

Khi giao tiếp vi điều khiển với nhiều phím nhấn, để tiết kiệm số chân port, ta sẽ kết nối các phím thành dạng ma trận, thay vì mỗi phím kết nối vào 1 chân port như ở chương 2.

Trên bàn phím ma trận, các phím nhấn được tổ chức thành dạng ma trận, ở đó một đầu của phím nối vào các hàng, đầu còn lại nối vào các cột như :

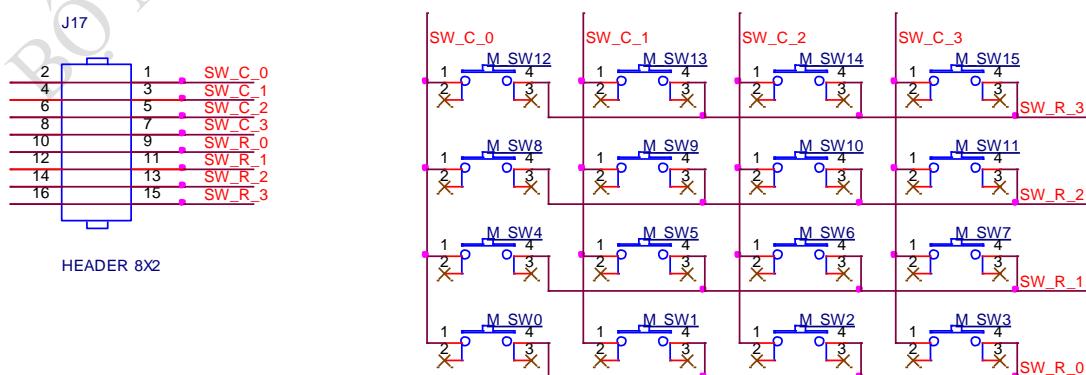


Hình 14 Kết nối bàn phím ma trận 4x4

Với cách kết nối trên, khi ta nhấn một phím thì hàng và cột sẽ được kết nối với nhau. Từ số hàng và cột trên ta có thể suy ra vị trí phím được nhấn.

3.2 GIAO TIẾP BÀN PHÍM MA TRẬN

Trên kit thí nghiệm, bàn phím ma trận gồm có 16 phím được tổ chức thành 4 hàng, bốn cột và kết nối tới header J17.



HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Hình 15 Sơ đồ khối bàn phím ma trận

Để giao tiếp bàn phím ma trận, ta kết nối header J17 đến 1 port của AVR.

Ta sử dụng kỹ thuật quét phím để nhận biết có phím nào được nhấn hay không. Bàn phím có thể được quét theo hàng hoặc theo cột. Sau đây trình bày quá trình quét phím theo cột.

Kết nối header J17 vào một port của AVR, ví dụ là PORTC. Khi đó, các cột SW_C3 đến SW_C0 được kết nối vào PC.3 đến PC.0, các hàng SW_R3 đến SW_R0 được kết nối đến PC.7 đến PC.4.

Để quét theo cột, ta cấu hình các chân giao tiếp với cột thành output và các chân giao tiếp với hàng thành input.

Cho cột 0 bằng 0 và 3 cột còn lại lên 1. Nếu như có bất kỳ phím nào trên cột 0 được nhấn, hàng tương ứng sẽ được kết nối vào cột 0 và sẽ có giá trị là 0. Như vậy ở lần quét này ta biết được có phím nào trên cột 0 được nhấn.

Tương tự, Cho cột 1 bằng 0 và 3 cột còn lại lên 1. Nếu như có bất kỳ phím nào trên cột 1 được nhấn, hàng tương ứng sẽ được kết nối vào cột 1 và sẽ có giá trị là 0. Như vậy ở lần quét này ta biết được có phím nào trên cột 1 được nhấn.

Làm tương tự đối với cột 2 và cột 3.

Như vậy, một chu kỳ quét phím sẽ quét qua lần lượt 4 cột, và phát hiện được có phím nhấn trên toàn bộ các phím trên ma trận phím.

Thời gian quét hết 4 cột là rất nhanh so với tốc độ nhấn và nhả phím của tay người, vì vậy đảm bảo khi bất kỳ phím nào được nhấn, quá trình quét phím sẽ phát hiện được.

Sau đây là chương trình con quét bàn phím ma trận được kết nối vào PORT A. Các cột từ C0 đến C3 kết nối vào PA.0 đến PA.3, các hàng từ R0 đến R3 kết nối vào PA4 đến PA7.

```
; ATmega324PA keypad scan function
; Scans a 4x4 keypad connected to PORTA
;C3-C0 connect to PA3-PA0
;R3-R0 connect to PA7-PA4
; Returns the key value (0-15) or 0xFF if no key is pressed

keypad_scan:
    ldi r20, 0b00001111 ; set upper 4 bits of PORTD as input with pull-up, lower 4
    bits as output
    out DDRA, r20
    ldi r20, 0b11111111 ; enable pull up resistor
    out PORTA, r20

    ldi r22, 0b11110111 ; initial col mask
    ldi r23, 0 ; initial pressed row value
    ldi    r24,3 ;scanning col index
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
keypad_scan_loop:  
    out PORTA, r22 ; scan current col  
    nop             ;need to have 1us delay to stablize  
    sbic PINA, 4 ; check row 0  
    rjmp keypad_scan_check_col2  
    rjmp keypad_scan_found ; row 0 is pressed  
keypad_scan_check_col2:  
    sbic PINA, 5 ; check row 1  
    rjmp keypad_scan_check_col3  
    ldi r23, 1 ; row1 is pressed  
    rjmp keypad_scan_found  
keypad_scan_check_col3:  
    sbic PINA, 6 ; check row 2  
    rjmp keypad_scan_check_col4  
    ldi r23, 2 ; row 2 is pressed  
    rjmp keypad_scan_found  
keypad_scan_check_col4:  
    sbic PINA, 7 ; check row 3  
    rjmp keypad_scan_next_row  
    ldi r23, 3 ; row 3 is pressed  
    rjmp keypad_scan_found  
  
keypad_scan_next_row:  
    ; check if all rows have been scanned  
    cpi r24,0  
    breq keypad_scan_not_found  
  
    ; shift row mask to scan next row  
    ror r22  
    dec r24           ;increase row index  
    rjmp keypad_scan_loop  
  
keypad_scan_found:  
    ; combine row and column to get key value (0-15)  
    ;key code = row*4 + col  
    lsl r23 ; shift row value 4 bits to the left  
    lsl r23  
    add r23, r24 ; add row value to column value  
    ret  
  
keypad_scan_not_found:  
    ldi r23, 0xFF ; no key pressed  
    ret
```

Ví dụ 8 Chương trình assembly quét phím

```
//ATmega324PA keypad scan function  
//Scans a 4x4 keypad connected to PORTA  
//C3-C0 connect to PA3-PA0  
//R3-R0 connect to PA7-PA4  
//Returns the key value (0-15) or -1 if no key is pressed  
  
int8_t keypad_scan() {  
  
    uint8_t col_mask = 0b11110111; // initial row mask  
    int8_t scan_col;           //scanning col index  
  
    DDRA = 0x0F; // set upper 4 bits of PORTA as input with pull-up, lower 4 bits  
as output  
    PORTA = 0xFF;//enable pullup resistor on the input pin
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
for (scan_col=3; scan_col>=0; scan_col--) {  
    PORTA = col_mask; // scan current col  
    _delay_us(1); // wait for stable input  
  
    if ((PINA & (1 << PINA4)) == 0) { // check row 0  
        return (scan_col) ; // calculate key value as (row index * 4)  
+ column index  
    }  
    else if ((PINA & (1 << PINA5)) == 0) { // check row 1  
        return (4 + scan_col) ; // calculate key value as (row index *  
4) + column index  
    }  
    else if ((PINA & (1 << PINA6)) == 0) { // check row 2  
        return (8 + scan_col) ; // calculate key value as (row index *  
4) + column index  
    }  
    else if ((PINA & (1 << PINA7)) == 0) { // check row 3  
        return (12 + scan_col) ; // calculate key value as (row index  
* 4) + column index  
    }  
  
    col_mask = ((col_mask >> 1) | (1<<7)) ; // shift row mask to scan next  
row  
}  
  
return -1;  
}
```

Ví dụ 9 Chương trình con quét phím dùng C

3.3 SỬ DỤNG KHỐI CÔNG AND ĐỂ TẠO NGẮT KHI CÓ PHÍM NHẤN

Để biết được có phím nhấn và xử lý, ta có thể quét liên tục phím nhấn. Đoạn chương trình sau sẽ quét phím và xuất ra Bar Led mã phím, sử dụng các chương trình con keypad_scan và shiftoutdata.

```
call initport  
call cleardata  
main:  
    call keypad_scan  
    mov     shiftData, r23  
  
    call shiftoutdata  
    jmp     main
```

Tuy nhiên, có thể thấy là CPU phải thực hiện quét phím một cách liên tục để không bị bỏ lỡ bất kỳ sự kiện phím nhấn nào. Điều này sẽ làm tiêu tốn thời gian hoạt động của CPU, không còn thời gian cho các tác vụ khác. Mặt khác, khi CPU đang thực hiện một tác vụ chiếm nhiều thời gian, có thể gây ra hiện tượng không đáp ứng với phím, vì khi đó chương trình quét phím không được gọi.

Để giải quyết vấn đề trên, ta có thể thiết kế hệ thống để sinh ra một ngắt khi có sự nhấn/nhả của phím. Chương trình quét phím sẽ được gọi ngay khi có ngắt.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Kit thí nghiệm có 2 cổng AND, được thiết kế như ở Hình 16. Để thí nghiệm, ta sử dụng một bus dây kết nối header J17 của khói bàn phím vào header J55, và dùng 1 bus dây khác kết nối J53 đến một port của AVR để quét phím.

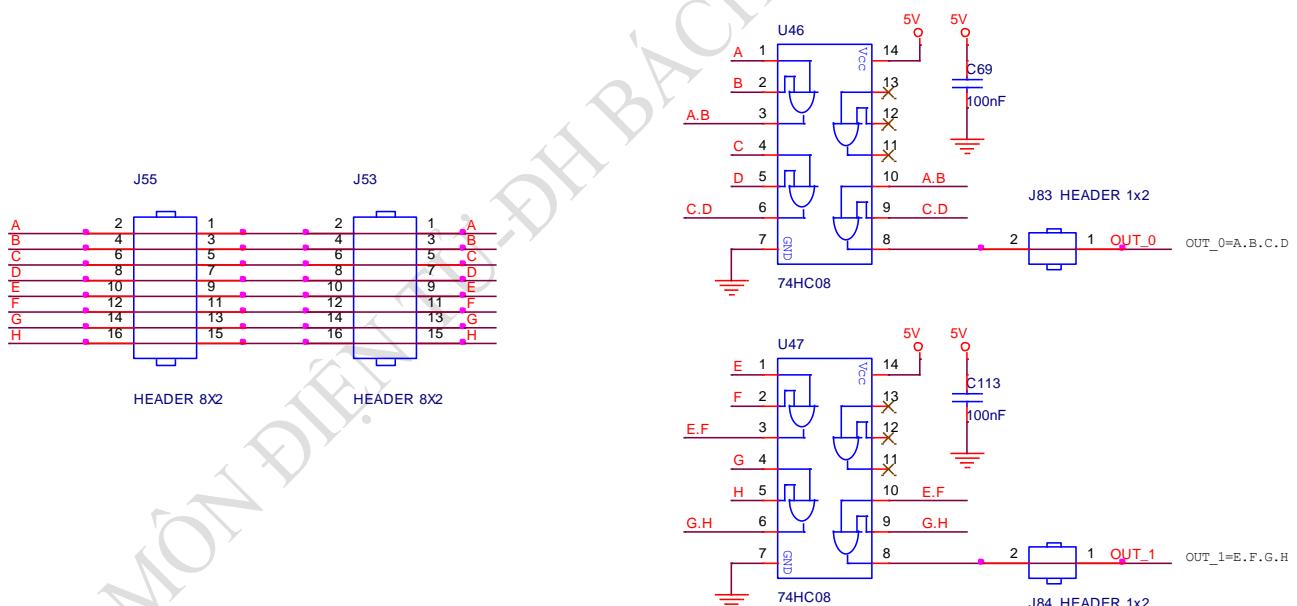
Khi đó, các tín hiệu C0-C3 sẽ kết nối đến các tín hiệu A, B, C, D và R0-R3 kết nối đến các tín hiệu E, F, G, H của khói cổng AND. Nếu ta quét phím theo cột, các tín hiệu R0-R3 sẽ là input, tín hiệu OUT1=R0&R1&R2&R3.

Kết nối tín hiệu OUT1 đến 1 chân Port có hỗ trợ ngắt ngoài (PD3, PD2, PB2). Cầu hình ngắt để kích hoạt theo mức thấp nếu ta muốn quét phím liên tục trong thời gian phím nhấn. Nếu ta chỉ muốn quét một lần khi phím nhấn/nhả, cầu hình ngắt ngoài theo cạnh xuống.

Bình thường, ta xuất mức 0 ra các tín hiệu cột COL3..0. Khi không có phím được nhấn, các tín hiệu R3..R0 sẽ bằng 1 do có điện trở kéo lên, khi đó OUT1 bằng 1.

Khi có một phím bất kỳ được nhấn, một trong số các tín hiệu R3..R0 sẽ được kéo xuống 0, làm tín hiệu OUT1 bằng 0 và kích hoạt ngắt.

Chương trình sẽ chuyển hướng đến trình phục vụ ngắt. Sau khi quét phím trong ngắt và xử lý,



Hình 16 Khối cổng AND

Lưu ý rằng các chân port của Atmega324 có hỗ trợ ngắt PINCHANGE, ta có thể sử dụng các ngắt này mà không cần phải dùng đến cổng AND.

Đoạn chương trình sau quét phím và hiển thị lên bar led sử dụng ngắt PINCHANGE trên các chân PA4, PA5, PA6, PA7

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
.equ BUTTON_MASK = (1<<PA4)|(1<<PA5)|(1<<PA6)|(1<<PA7) ; mask for checking button state

.org 0x0000 ; interrupt vector table
rjmp reset_handler ; reset

.org 0x001C ; pin change interrupt vector
rjmp pinchange_handler

reset_handler:
    ; initialize stack pointer
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16
    call initport
    call cleardata
    call keypad_prepare_port
    ; enable pin change interrupt for PA3, PA2, PA1, and PA0
    ldi r16, BUTTON_MASK
    sts PCMSK0, r16
    ldi r16, (1<<PCIE0)
    sts PCICR, r16
    ; enable global interrupts
    sei

main:
    jmp main
; Set PA7..PA4 as input, PA3..PA0 as output
;set PA3..PA0 to 0
keypad_prepare_port:
    ldi r20, 0b00001111 ; set upper 4 bits of PORTD as input with pull-up, lower 4
;bits as output
    out DDRA, r20
    ldi r20, 0b11110000 ; enable pull up resistor
    out PORTA, r20
    ret

pinchange_handler:
    ; check if PA7, PA6, PA5, or PA4 has changed
    in r16, PINA
    andi r16, BUTTON_MASK
    breq pinchange_handler_exit ; exit if none of the buttons have changed

    call keypad_scan ;scan and shiftout keycode
    mov shiftData,r23
    call shiftoutdata

pinchange_handler_exit:
    call keypad_prepare_port
    reti ; return from interrupt
```

Ví dụ 10 Quét phím và hiển thị dùng ngắt PORTCHANGE (Assembly)

```
#define BUTTON_MASK ((1 << PA7) | (1 << PA6) | (1 << PA5) | (1 << PA4))
void portIntPrepare()
{
    DDRA = 0x0F; // set upper 4 bits of PORTA as input with pull-up, lower
4 bits as output
    PORTA = 0xF0;//enable pullup resistor on the input pin, clear 4 lower
bit
    return;
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
}

void portchangeIntEna()
{
    // enable pin change interrupt for PA7, PA6, PA5, and PA4
    PCMSK0 |= BUTTON_MASK;
    PCICR |= (1 << PCIE0);
    // enable global interrupts
    sei();
}

int main() {

    initializePorts();
    portIntPrepare();
    clearShiftRegister();
    portchangeIntEna();
    while (1) {
        //shiftOut(keypad_scan());
    }
    return 0;
}
ISR(PCINT0_vect) {
    shiftOut(keypad_scan());
    portIntPrepare();
}
```

Ví dụ 11 Quét phím và hiển thị dùng ngắt PORTCHANGE (C)

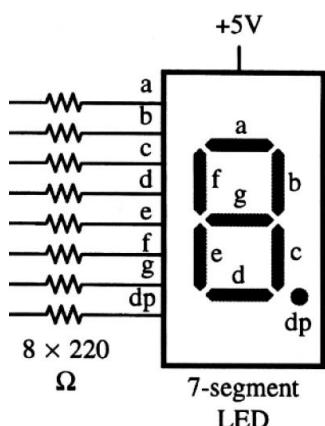
BỘ MÔN ĐIỆN TỬ - ĐH BÁCH KHOA TP.HCM

CHƯƠNG 4 HIỂN THỊ DÙNG LED 7 ĐOẠN

4.1 LÝ THUYẾT CƠ BẢN

Led 7 đoạn thường được dùng để hiển thị các chữ số hoặc chữ cái đơn giản. Giao tiếp này cho phép người không rành về kỹ thuật cũng có thể sử dụng hệ thống thông qua việc đọc các thông tin hiển thị trên led. Ví dụ nhiều led 7 đoạn có thể được dùng để hiển thị số điện thoại tại các buồng gọi điện thoại công cộng, hoặc các giá trị giây tại các trục đèn giao thông.

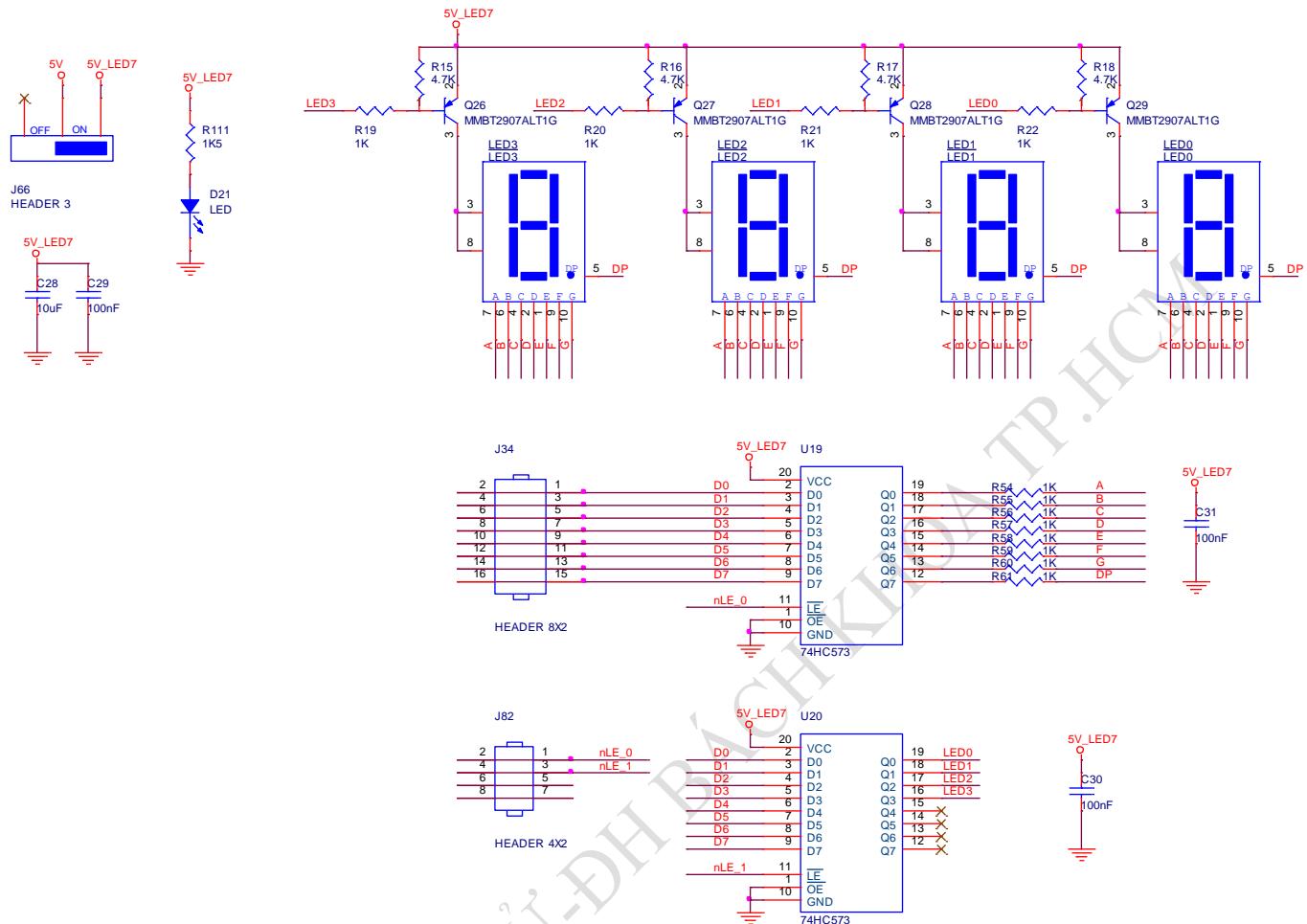
Cấu tạo của led 7 đoạn gồm 8 led đơn được nối chung cực anode (dạng led anode chung) hoặc nối chung cực cathode (dạng led cathode chung). Các đoạn led đơn này được đặt tên là a, b, c, d, e, f, g và dấu chấm dp. Để hiển thị một giá trị lên led 7 đoạn, cần cấp điện áp lên chân chung và các tín hiệu điều khiển đoạn tương tự như điều khiển các led đơn.



Hình 2 Led 7 đoạn dạng anode chung

Để hiển thị một giá trị số lên led 7 đoạn, người lập trình cần xuất các giá trị để điều khiển các led a, b,..., g và dp. Tuy nhiên, dữ liệu trong các hệ thống vi xử lý thường tồn tại dưới dạng nhị phân, dạng này không thể trực tiếp hiển thị lên led 7 đoạn. Do đó cần phải thực hiện chuyển đổi từ mã biểu diễn nhị phân sang mã biểu diễn lên led 7 đoạn. Việc chuyển đổi này có thể thực hiện bằng phần cứng với vi mạch chuyển mã hoặc dùng phần mềm (phương pháp tra bảng, look-up table).

4.2 THIẾT KẾ PHẦN CỨNG



Hình 17 Sơ đồ thiết kế khối led 7 đoạn

Khối led 7 đoạn gồm có 4 led anode chung, được thiết kế để hiển thị theo kiểu quét LED. Hai IC 74HC573 U19 và U20 có ngõ vào dữ liệu nối chung với nhau và nối vào header J34, các tín hiệu điều khiển chốt nLE0 và nLE1 trên header J82.

Các tín hiệu đoạn A đến G và DP của các led 7 đoạn được nối chung và nối vào ngõ ra của IC chốt 74HC573 (U19). IC chốt này được điều khiển bởi tín hiệu nLE0 trên header J82. Để các đoạn sáng, phải xuất mức 0 ra các chân đoạn tương ứng, và xuất mức 1 ra các chân đoạn tối.

Các led 7 đoạn được phép hiển thị bằng cách mở khóa BJT cấp nguồn cho led. Tín hiệu mức 0 tại cực B của các BJT Q25, Q26, Q27, Q28 sẽ làm cho BJT dẫn bão hòa và cực C của BJT sẽ có giá trị gần 5V (gần bằng điện thế tại cực E) cung cấp nguồn 5V cho led 7 đoạn. Các tín hiệu mở khóa BJT LED3, LED2, LED1, LED0 nối vào ngõ ra của IC chốt U20. IC này được điều khiển bởi tín hiệu nLE1 trên header J82.

Khối LED được cấp nguồn thông qua Jumper J66. Trước khi lập trình, set vị trí Jumper sang vị trí ON để cấp nguồn cho khói LED 7 đoạn.

4.3 HIỂN THỊ LÊN LED 7 ĐOẠN

4.3.1 Hiển thị lên 1 LED:

Như đã thấy ở thiết kế phần cứng, để có thể hiển thị một số lên led 7 đoạn, người lập trình cần xuất mã 7 đoạn của số hay ký tự cần hiển thị ra ngõ vào của vi mạch ‘573 U19. Để dữ liệu này được chốt ra ngõ ra của IC 74HC573, đầu tiên ta đưa tín hiệu nLE0 lên 1 để đưa dữ liệu ra, sau đó đưa tín hiệu này xuống 0 để chốt lại. Sau đó, ta cho dữ liệu thích hợp chốt vào U23 bằng cách đặt dữ liệu lên header J34 và tích cực tín hiệu nLE1.

INPUTS			OUTPUTQ
\overline{OE}	LE	D	
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

Bảng 2 Bảng sự thật của IC 74HC573

Giả sử ta muốn hiển thị số 1 lên LED số 3. Đầu tiên, ta phải xuất mã 7 đoạn của số 1 là F9 ra IC U19, sau đó bật LED 3 bằng cách xuất giá trị 0x07 ra IC U23 để tích cực LED3.

Số	Số nhị phân								HEX
	7	6	5	4	3	2	1	0	
	d	p	g	f	e	d	c	b	a
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	8F
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90
A	1	0	0	0	1	0	0	0	88
B	1	0	0	0	0	0	1	1	83
C	1	1	0	0	0	1	1	0	C6
D	1	0	1	0	0	0	0	1	A1
E	1	0	0	0	0	1	1	0	86
F	1	0	0	0	1	1	1	0	8E

Bảng 3 Bảng mã 7 đoạn cho LED anode chung

Đoạn chương trình sau khởi động chân port và xuất giá trị lên LED 7 đoạn. J34 kết nối vào PORTD, nLE0 kết nối vào PB4, nLE1 kết nối vào PB5

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
; Lookup table for 7-segment codes
table_7seg_data:    .DB 0XC0, 0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,0X8
                   .DB           0XC6,0XA1,0X86,0X8E

; Lookup table for LED control
table_7seg_control:
                   .DB          0b00001110,0b00001101, 0b00001011, 0b00000111

; J34 connect to PORTD
; nLE0 connect to PB4
; nLE1 connect to PB5

; Output: None

.equ   LED7SEGPORT = PORTD
.equ   LED7SEGDIR = DDRD
.equ   LED7SEGLatchPORT = PORTB
.equ   LED7SEGLatchDIR = DDRB
.equ   nLE0Pin          =      4
.equ   nLE1Pin          =      5

led7seg_portinit:
    push   r20
    ldi    r20, 0b11111111 ; SET led7seg PORT as output
    out   LED7SEGDIR, r20
    in    r20, LED7SEGLatchDIR ; read the Latch Port direction register
    ori   r20, (1<<nLE0Pin) | (1 << nLE1Pin)
    out   LED7SEGLatchDIR,r20
    pop   r20
    ret

; Display a value on a 7-segment LED using a lookup table
; Input: R27 contains the value to display
;        R26 contain the LED index (3..0)
;        J34 connect to PORTD
;        nLE0 connect to PB4
;        nLE1 connect to PB5
; Output: None

display_7seg:
    push  r16 ; Save the temporary register

    ; Look up the 7-segment code for the value in R18
    ; Note that this assumes a common anode display, where a HIGH output turns OFF
    ; the segment
    ; If using a common cathode display, invert the values in the table above
    ldi    zh,high(table_7seg_data<<1) ;
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
ldi      zl,low(table_7seg_data<<1) ;  
clr      r16  
add      r30,r27  
adc      r31,r16  
  
lpm      r16,z  
out      LED7SEGPORTR,r16  
sbi      LED7SEGLatchPORT,nLE0Pin  
nop  
cbi      LED7SEGLatchPORT,nLE0Pin  
ldi      zh,high(table_7seg_control<<1);  
ldi      zl,low(table_7seg_control<<1);  
clr      r16  
add      r30,r26  
adc      r31,r16  
lpm      r16,z  
out      LED7SEGPORTR,r16  
sbi      LED7SEGLatchPORT,nLE1Pin  
nop  
cbi      LED7SEGLatchPORT,nLE1Pin  
pop      r16; Restore the temporary register  
ret ; Return from the function
```

Ví dụ 12 Chương trình hiển thị LED 7 đoạn dùng assembly

```
#define LED7SEGPORTR PORTD  
define LED7SEGDIRR DDRD  
  
define LED7SEGLatchPORTR PORTB  
define LED7SEGLatchDIRR DDRB  
  
define nLE0Pin          4  
define nLE1Pin          5  
  
const uint8_t led7seg_code[16] = {0XC0,  
0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,0X8,0XC6,0XA1,0X86,0X8E};  
const uint8_t led7seg_control[4] = {0b00000111,0b00001011,0b00001101, 0b00001110};  
  
void led7segInitializePorts() {  
    // Set output pins  
    LED7SEGDIRR = 0xFF;  
    LED7SEGLatchDIRR |= (1<<nLE0Pin) | (1 << nLE1Pin);  
    return;  
}  
void led7segDisplay(uint8_t      value, uint8_t ledIndex)  
{  
    LED7SEGPORTR = led7seg_code[value];  
    LED7SEGLatchPORTR |= (1<<nLE0Pin);  
    LED7SEGLatchPORTR &= ~(1<<nLE0Pin);
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
LED7SEGPORT = led7seg_control[ledIndex];
LED7SEGLatchPORT |= (1<<nLE1Pin);
LED7SEGLatchPORT &= ~(1<<nLE1Pin);
return;
}
```

Ví dụ 13 Chương trình hiển thị LED 7 đoạn dùng C

4.3.2 Hiển thị đồng thời lên các LED 7 đoạn

Thiết kế của kit thí nghiệm không cho phép hiển thị 4 số khác nhau lên cả 4 led một cách đồng thời. Để có thể nhìn thấy 4 giá trị cùng 1 lúc ta dùng phương pháp quét LED.

Ví dụ như ta cần hiển thị 4 số 1234 ra 4 LED. Để làm được điều này, ta sẽ cho LED0 hiển thị số 4 trong một thời gian T, sau đó tắt LED0 và cho LED1 hiển thị số 3 cũng trong thời gian T. Tương tự cho LED2 và LED3.

Như vậy, các LED sẽ lờn lượt sáng/tắt. Trong 1 giây số lần sáng của mỗi LED sẽ là $1s/4T$.

Nhờ vào hiệu ứng lưu ảnh của mắt người, nếu LED sáng tắt đủ nhanh thì mắt người sẽ không nhận ra hiện tượng chớp nháy. Theo lý thuyết, nếu số lần LED sáng tắt lớn hơn 24 lần trong 1 giây thì mắt người coi như LED sáng liên tục. Như vậy, thời gian T càng nhỏ thì mắt người càng thấy hình ảnh ổn định.

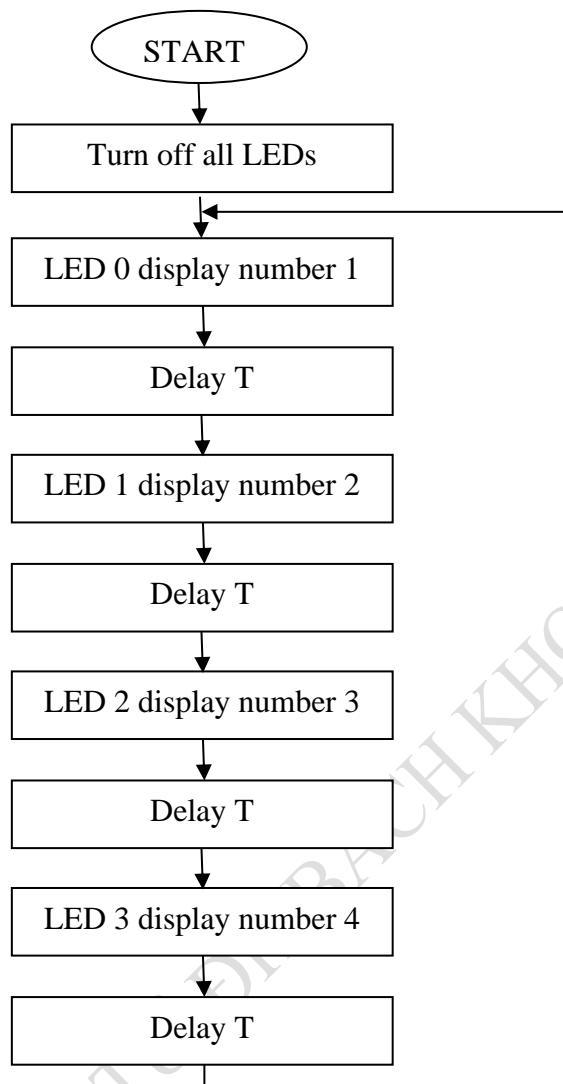
Trên kit thí nghiệm có 4 LED, nếu chọn tần số quét là 25HZ thì thời gian sáng của mỗi LED sẽ là 10ms.

Đoạn chương trình sau sẽ hiển thị số 1234 lên 4 LED 7 đoạn trên kit thí nghiệm.

Trên thực tế, với thời gian sáng cho mỗi LED là 10ms, nghĩa là tần số quét là 25Hz, ta vẫn nhận thấy sự nhấp nháy của LED. Thông thường, để LED sáng ổn định, ta sẽ quét LED với tần số 50HZ.

```
clr    r26          ;start index = 0
ldi    r27,1        ;start value = 1
main:
    call   display_7seg
    call   DELAY_10MS
    inc    r26
    inc    r27
    andi  r26,0x0f
    cpi   r26,4
    brne main
    ldi   r27,1
    clr   r26
    jmp   main
```

Ví dụ 14 Quét LED sử dụng vòng lặp delay



Hình 18 Lưu đồ chương trình quét LED

4.4 QUÉT LED SỬ DỤNG NGẮT TIMER

Cách quét LED dựa vào các hàm delay để tạo trễ như trong phần 4.3.2 có ưu điểm là đơn giản, tuy nhiên có nhược điểm lớn là toàn bộ thời gian của CPU được sử dụng để tạo trễ, không còn tài nguyên để thực hiện các tác vụ khác. Nếu ta chèn các tác vụ khác vào trong vòng lặp hiển thị LED, định thời của quá trình hiển thị sẽ bị thay đổi và làm cho các LED sáng không đều, thậm chí nháy nháy.

Để giải quyết vấn đề này, ta sẽ sử dụng ngắt timer để sinh ra ngắt sau mỗi khoảng thời gian phù hợp. Khi xảy ra ngắt, ta sẽ thực hiện xuất dữ liệu và bật LED tương ứng.

Chương trình ví dụ sau thực hiện việc hiển thị 4 giá trị chứa trong các ô nhớ ở địa chỉ 0x0100, 0x0101, 0x0102, 0x0103 ra 4 LED 7 đoạn trên kit.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Timer 1 được cấu hình để tạo ra ngắt sau mỗi 10 ms. Khi ngắt đầu tiên xảy ra, LED3 sẽ hiển thị giá trị ở ô nhớ 0x0100, 3 LED còn lại tắt. Khi xảy ra ngắt lần thứ 2, LED2 sẽ hiển thị giá trị ở ô nhớ 0x0101. Quá trình sẽ diễn ra tương tự cho 2 LED còn lại và lặp đi lặp lại.

Số đếm lưu vị trí hiện tại của LED chứa trong ô nhớ **LED7segIndex**. Số đếm này sẽ được cập nhật tự động trong quá trình quét.

Với cách thực hiện này, tổng thời gian CPU phải dùng cho việc quét LED là rất nhỏ, và đảm bảo các LED được hiển thị đúng theo định thì yêu cầu.

Giá trị hiển thị trên LED được thay đổi bằng cách ghi giá trị vào 4 ô nhớ trong RAM tại **LED7segValue**. Các ô nhớ này đóng vai trò như một “frame buffer” cho việc hiển thị.

Ví dụ ta muốn hiển thị số 1-2-3-4 lên 4 LED, ta lần lượt ghi giá trị 1-2-3-4 vào 4 ô nhớ liên tiếp nhau bắt đầu từ **LED7segValue**.

```
.include "m324padef.inc" ; Include Atmega324pa definitions
.org 0x0000 ; interrupt vector table
rjmp reset_handler ; reset
.org 0x001A
rjmp timer1_COMP_ISR
reset_handler:
    ; initialize stack pointer
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, (1<<PCIE0)
    sts PCICR, r16
    call initTimer1CTC
    call    led7seg_portinit
    call    Led7seg_buffer_init

    ; enable global interrupts
    sei
main:
    jmp      main
; Lookup table for 7-segment codes
table_7seg_data:
    .DB      0XC0,0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,0X83
    .DB      0XC6,0XA1,0X86,0X8E
; Lookup table for LED control
table_7seg_control:
    .DB      0b00001110,0b00001101, 0b00001011, 0b00000111
;           J34 connect to PORTD
;           nLE0 connect to PB4
;           nLE1 connect to PB5
; Output: None

.equ LED7SEGPORT = PORTD
.equ LED7SEGDIR = DDRD
.equ LED7SEGLatchPORT = PORTB
.equ LED7SEGLatchDIR = DDRB
.equ nLE0Pin          =      4
.equ nLE1Pin          =      5
.dseg
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
.org    SRAM_START           ;starting address is 0x100
        LED7segValue: .byte 4 ;store the BCD value to display
        LED7segIndex: .byte 1

.cseg
.align 2
;init the Led7seg buffer
Led7seg_buffer_init:
    push    r20
    ldi     r20,3
    ldi     r31,high(LED7segIndex)
    ldi     r30,low(LED7segIndex)
    st      z,r20
    ldi     r20,0
    ldi     r31,high(LED7segValue)
    ldi     r30,low(LED7segValue)
    st      z+,r20          ;LED index start at 3
    inc    r20
    st      z+,r20
    inc    r20
    st      z+,r20
    inc    r20
    st      z+,r20
    pop    r20
    ret

led7seg_portinit:
    push   r20
    ldi    r20, 0b11111111 ; SET led7seg PORT as output
    out   LED7SEGDIR, r20
    in    r20, LED7SEGLatchDIR ; read the Latch Port direction register
    ori   r20, (1<<nLE0Pin) | (1 << nLE1Pin)
    out   LED7SEGLatchDIR,r20
    pop   r20
    ret;

;Display a value on a 7-segment LED using a lookup table
; Input: R27 contains the value to display
;         R26 contain the LED index (3..0)
;         J34 connect to PORTD
;         nLE0 connect to PB4
;         nLE1 connect to PB5
; Output: None
display_7seg:
    push  r16 ; Save the temporary register

    ; Look up the 7-segment code for the value in R18
    ; Note that this assumes a common anode display, where a HIGH output turns OFF
the segment
    ; If using a common cathode display, invert the values in the table above
    ldi     zh,high(table_7seg_data<<1) ;
    ldi     zl,low(table_7seg_data<<1) ;
    clr    r16
    add    r30, r27
    adc    r31,r16

    lpm    r16, z
    out   LED7SEGPORT,r16
    sbi   LED7SEGLatchPORT,nLE0Pin
    nop
    cbi   LED7SEGLatchPORT,nLE0Pin
    ldi     zh,high(table_7seg_control<<1) ;
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

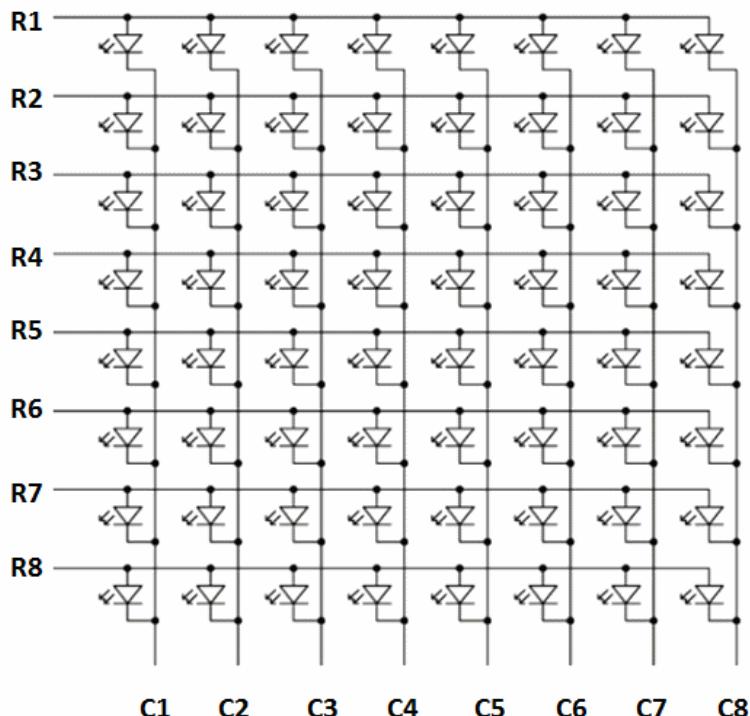
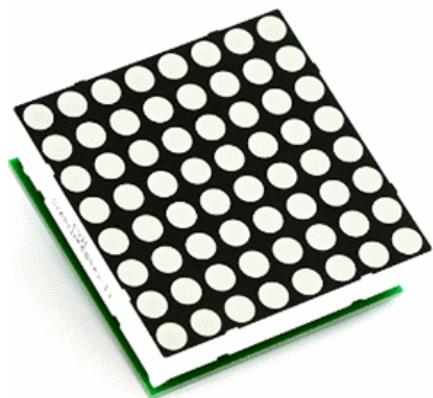
```
ldi      z1,low(table_7seg_control<<1)      ;
clr      r16
add      r30, r26
adc      r31,r16
lpm      r16, z
out      LED7SEGPORT,r16
sbi      LED7SEGLatchPORT,nLE1Pin
nop
cbi      LED7SEGLatchPORT,nLE1Pin
pop      r16 ; Restore the temporary register
ret ; Return from the function

initTimer1CTC:
    push r16
    ldi r16, high(10000) ; Load the high byte into the temporary register
    sts OCR1AH, r16       ; Set the high byte of the timer 1 compare value
    ldi r16, low(10000)   ; Load the low byte into the temporary register
    sts OCR1AL, r16       ; Set the low byte of the timer 1 compare value
    ldi r16, (1 << CS10)| (1<< WGM12) ; Load the value 0b00000101 into the
temporary register
    sts TCCR1B, r16       ;
    ldi r16, (1 << OCIE1A); Load the value 0b00000010 into the temporary
register
    sts TIMSK1, r16        ; Enable the timer 1 compare A interrupt
    pop r16
    ret

timer1_COMP_ISR:
    push r16
    push r26
    push r27
    ldi      r31,high(LED7segIndex)
    ldi      r30,low(LED7segIndex)
    ld       r16,z
    mov      r26,r16
    ldi      r31,high(LED7segValue)
    ldi      r30,low(LED7segValue)
    add      r30,r16
    clr      r16
    adc      r31,r16
    ld       r27,z
    call     display_7seg

    cpi      r26,0
    brne   timer1_COMP_ISR_CONT
    ldi      r26,4 ;if r16 = 0, reset to 3
timer1_COMP_ISR_CONT:
    dec      r26          ;else, decrease
    ldi      r31,high(LED7segIndex)
    ldi      r30,low(LED7segIndex)
    st       z,r26

    pop      r27
    pop      r26
    pop      r16
    reti
```

CHƯƠNG 5 LED MA TRẬN**5.1 LÝ THUYẾT CƠ BẢN****Hình 19 Cấu trúc LED ma trận**

Led ma trận là tập hợp các led được bố trí theo dạng ma trận. Trên Hình 19 là 1 LED ma trận 8x8 đơn sắc, gồm có 64 LED được bố trí thành 8 hàng và 8 cột. Các LED trên 1 hàng có anode nối chung với nhau, các LED trên 1 cột có cathode nối chung với nhau. Để thắp sáng 1 LED ở hàng R và cột C, ta đưa điện áp dương vào hàng R và điện áp âm (hoặc GND) vào cột C.

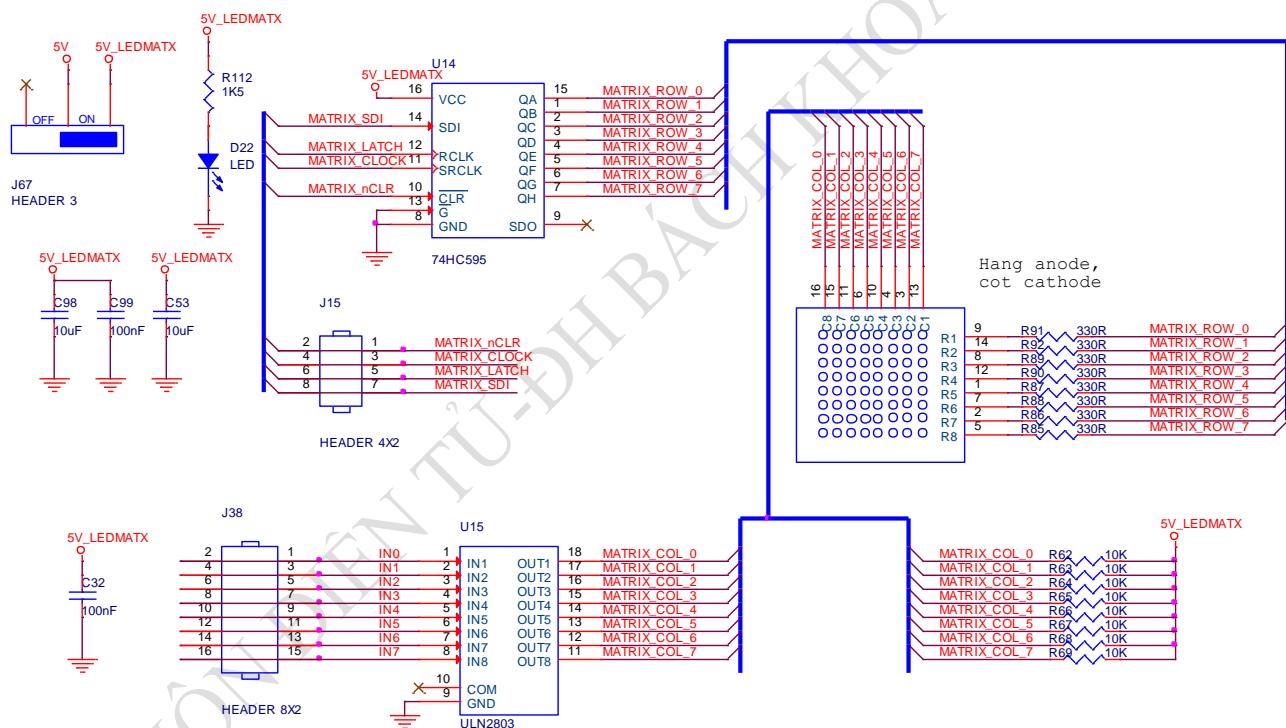
Led ma trận có thể dùng để hiển thị thông tin đến từng điểm ảnh. Do đó, led ma trận có thể biểu diễn được chữ số, chữ cái, và các hình ảnh khác. Led ma trận có thể là dạng đơn sắc (thường là màu đỏ), hoặc đa sắc. Đối với dạng đa sắc, một điểm ảnh chứa đến 2 led với màu khác nhau (ví dụ màu đỏ và màu xanh lá cây). Sử dụng phối hợp hai màu này có thể sinh ra thêm một số màu trung gian. Quá trình này có thể được thực hiện bằng cách thay đổi độ rộng xung điều khiển của từng màu (tức là thời gian sáng của từng led màu). Đối với led ma trận, cả hai chân anode và cathode của led đều được đưa ra thành tín hiệu điều khiển.

5.2 THIẾT KẾ PHẦN CỨNG

Led ma trận trên kit là led có kích thước 8x8. Các tín hiệu hàng (ROW0-ROW7) được kết nối đến ngõ ra của 1 thanh ghi dịch 74HC595. Các cột được kết nối đến các ngõ ra của IC ULN2803.

Thiết kế sử dụng vi mạch ULN2803 cho phép kéo dòng lên đến 500mA ở phía cột. Để hiển thị một điểm ảnh thì dữ liệu xuất ra trên hàng phải là mức 1, cấp nguồn VCC cho anode của LED. Cột tương ứng sẽ được kéo xuống GND bằng cách xuất logic 1 vào ngõ vào tương ứng của IC ULN2803.

Ví dụ ta muốn LED ở hàng 0 và cột 0 sáng, ta sẽ cho tín hiệu MATRIX_ROW_0 bằng 1 và tín hiệu IN0 bằng 1 (làm cho tín hiệu MATRIX_COL0 bằng 0).



Hình 20 Thiết kế khối LED ma trận

5.3 LẬP TRÌNH GIAO TIẾP LED MA TRẬN

Tương tự như LED 7 đoạn, để có thể hiển thị lên led ma trận, ta phải hiển thị dữ liệu lần lượt theo từng cột.

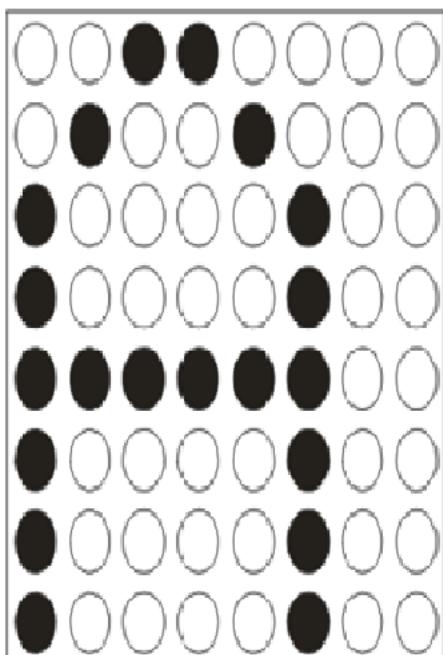
Mỗi cột sẽ được hiển thị trong một khoảng thời gian nhất định sau đó chuyển sang cột kế tiếp. Cũng tương tự như LED 7 đoạn, các cột phải được quét ít nhất 25 lần trong 1s, tuy nhiên trong thực tế ta chọn tần số là 50Hz để đảm bảo LED không cảm thấy bị chớp.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Để hiển thị được 1 cột, người lập trình cần xuất 1 byte ra hàng (1 byte trong 8 byte tra được từ bảng font), cho phép cột đó, chờ một khoảng thời gian và chuyển sang hiển thị cột kế tiếp với qui trình tương tự. Sau khi hết 8 cột, qui trình hiển thị được lặp lại từ đầu.

Led ma trận được sử dụng là 8 cột, mỗi cột có 8 hàng. Dữ liệu để hiển thị trên mỗi cột sẽ gồm 8 bit, hay 1 byte. Vì vậy ta sẽ cần 8 byte để lưu thông tin muốn hiển thị.

Ví dụ ta muốn hiển thị ký tự chữ A lên led ma trận.



C0	C1	C2	C3	C4	C5	C6	C7
0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0
1	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0

Hình 3 Chữ A trên led ma trận 8x8

Bảng trên mô tả giá trị của 8 byte cần thiết để xuất chữ A. Đây có thể gọi là font của chữ A.

Các font này có thể được chứa trong bộ nhớ chương trình như sau:

ledmatrix_Font_A:

```
.DB    0b11111100,    0b00010010,    0b00010001,    0b00010001,    0b00010010,    0b11111100,  
0b00000000,    0b00000000
```

Ta có thể tạo ra dữ liệu cho các hình ảnh tùy ý muốn xuất ra trên LED ma trận bằng cách tương tự.

Để tạo ra font của các ký tự dễ dàng, ta có thể sử dụng phần mềm LCD Font Maker.

5.3.1 Kết nối phần cứng trên kit

- Kết nối mạch lái cột

Có nhiều cách kết nối phần cứng trên kit để thực hiện giao tiếp LED ma trận. Đơn giản nhất là kết nối header J38 (ngõ vào của ULN2803) vào 1 port của AVR dùng 1 dây bus, và kết nối các tín hiệu trên J15 vào các chân port khác để điều khiển thanh ghi dịch.

Một cách khác, ta có thể sử dụng thanh ghi dịch U9 ở khối 07-SHIFT REGISTER và kết nối ngõ ra của nó (header J35) vào J38. Khi đó ta điều khiển cả hàng và cột sử dụng 2 thanh ghi dịch để tiết kiệm chân port.

- Kết nối mạch lái hàng

Chọn 4 bit Port ,ví dụ PB0..PB3 lần lượt kết nối với các chân DI,SCLK,SRCLK,CLR của shift register U4 qua J15.

Sau khi kết nối, ta cấp nguồn cho khói LED ma trận bằng cách đưa jumper J67 sang vị trí ON.

5.3.2 Lập trình hiển thị lên LED ma trận

Đoạn chương trình sau hiển thị ký tự A với font đã được tạo như ở phần trước lên LED ma trận, sử dụng ngắt timer 1, với thời gian giữa hai lần tràn timer là 2500 clock.

Font được chứa trong ROM, và copy vào RAM tại địa chỉ LedMatrixBuffer.

```
.include "m324padef.inc" ; Include Atmega324pa definitions
.org 0x0000 ; interrupt vector table
rjmp reset_handler ; reset

.org 0x001A
rjmp timer1_COMP_ISR
reset_handler:
    ; initialize stack pointer
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16
    call shiftregister_initport
    call shiftregister_cleardata
    call initTimer1CTC
    ; enable global interrupts
    sei
    call ledmatrix_portinit
main:
    jmp main

.equ clearSignalPort = PORTB ; Set clear signal port to PORTB
.equ clearSignalPin = 3 ; Set clear signal pin to pin 3 of PORTB
.equ shiftClockPort = PORTB ; Set shift clock port to PORTB
.equ shiftClockPin = 2 ; Set shift clock pin to pin 2 of PORTB
.equ latchPort = PORTB ; Set latch port to PORTB
.equ latchPin = 1 ; Set latch pin to pin 1 of PORTB
.equ shiftDataPort = PORTB ; Set shift data port to PORTB
.equ shiftDataPin = 0 ; Set shift data pin to pin 0 of PORTB

; Initialize ports as outputs
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
shiftregister_initport:  
    push    r24  
    ldi     r24,  
(1<<clearSignalPin)|(1<<shiftClockPin)|(1<<latchPin)|(1<<shiftDataPin);  
    out     DDRB, r24           ; Set DDRB to output  
    pop    r24  
    ret  
  
shiftregister_cleardata:  
    cbi    clearSignalPort, clearSignalPin      ; Set clear signal pin to low  
; Wait for a short time  
    sbi    clearSignalPort, clearSignalPin      ; Set clear signal pin to high  
    ret  
; Shift out data  
;shift out R27 to bar led  
shiftregister_shiftoutdata:  
    push    r18  
    cbi    shiftClockPort, shiftClockPin        ;  
    ldi    r18, 8                 ; Shift 8 bits  
shiftloop:  
    sbrc   r27, 7      ; Check if the MSB of shiftData is 1  
    sbi    shiftDataPort, shiftDataPin          ; Set shift data pin to high  
    sbi    shiftClockPort, shiftClockPin        ; Set shift clock pin to high  
    lsl    r27            ; Shift left  
    cbi    shiftClockPort, shiftClockPin        ; Set shift clock pin to low  
    cbi    shiftDataPort, shiftDataPin          ; Set shift data pin to low  
    dec    r18  
    brne   shiftloop  
; Latch data  
    sbi    latchPort, latchPin      ; Set latch pin to high  
    cbi    latchPort, latchPin      ; Set latch pin to low  
    pop    r18  
    ret  
  
;Lookup table for column control  
ledmatrix_col_control: .DB 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01  
; Lookup table for font  
ledmatrix_Font_A:    .DB      0b11111100, 0b00010010, 0b00010001, 0b00010001,  
0b00010010, 0b11111100, 0b00000000, 0b00000000  
; J38 connect to PORTD  
; clear signal pin to pin 0 of PORTB  
; shift clock pin to pin 1 of PORTB  
; latch pin to pin 0 of PORTB  
; shift data pin to pin 3 of PORTB  
; Output: None  
  
.equ   LEDMATRIXPORT = PORTD  
.equ   LEDMATRIXDIR =  DDRD  
.dseg  
.org   SRAM_START           ;starting address is 0x100  
        LedMatrixBuffer   :     .byte 8  
        LedMatrixColIndex :     .byte 1  
.cseg  
.align 2  
ledmatrix_portinit:  
    push   r20  
    push   r21  
    ldi    r20, 0b11111111 ; SET port as output  
    out    LEDMATRIXDIR, r20  
  
    ldi    r20,0             ;col index start at 0
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
ldi      r31,high(LedMatrixColIndex)
ldi      r30,low(LedMatrixColIndex)
st      z,r20
ldi      r20,0
ldi      r31,high(ledmatrix_Font_A << 1) ;Z register point to fontA
value
ldi      r30,low(ledmatrix_Font_A << 1)
ldi      r29,high(LedMatrixBuffer) ; Y register point to fontA value
ldi      r28,low(LedMatrixBuffer)
ldi      r20,8
ledmatrix_portinit_loop:           ;copy font to display buffer
lpm      r21,z+
st      y+,r21
dec     r20
cpi     r20,0
brne   ledmatrix_portinit_loop
pop    r21
pop    r20
ret
;Display a Column of Led Matrix
;Input: R27 contains the value to display
;; R26 contain the Col index (3..0)
;Output: None
ledmatrix_display_col:
    push r16 ; Save the temporary register
    push r27
    clr   r16
    out   LEDMATRIXPORT,r16
    call  shiftregister_shiftoutdata

    ldi      r31,high(ledmatrix_col_control << 1)
    ldi      r30,low(ledmatrix_col_control << 1)
    clr   r16
    add   r30,r26
    adc   r31,r16
    lpm   r27,z
    out   LEDMATRIXPORT,r27
    pop    r27
    pop    r16 ; Restore the temporary register
    ret ; Return from the function

initTimer1CTC:
    push r16
    ldi r16, high(2500) ; Load the high byte into the temporary register
    sts OCR1AH, r16       ; Set the high byte of the timer 1 compare value
    ldi r16, low(2500)   ; Load the low byte into the temporary register
    sts OCR1AL, r16       ; Set the low byte of the timer 1 compare value
    ldi r16, (1 << CS10)| (1<< WGM12) ; Load the value 0b00000101 into the temporary register
    sts TCCR1B, r16       ;
    ldi r16, (1 << OCIE1A); Load the value 0b00000010 into the temporary register
    sts TIMSK1, r16        ; Enable the timer 1 compare A interrupt
    pop r16
    ret

timer1_COMP_ISR:
    push r16
    push r26
    push r27
    ldi      r31,high(LedMatrixColIndex)
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
ldi      r30,low(LedMatrixColIndex)
ld      r16,z
mov      r26,r16
ldi      r31,high(LedMatrixBuffer)
ldi      r30,low(LedMatrixBuffer)
add      r30,r16
clr      r16
adc      r31,r16
ld      r27,z
call    ledmatrix_display_col

inc      r26
cpi      r26,8
brne   timer1_COMP_ISR_CONT
ldi      r26,0 ;if r26 = 8, reset to 0
timer1_COMP_ISR_CONT:
ldi      r31,high(LedMatrixColIndex)
ldi      r30,low(LedMatrixColIndex)
st      z,r26

pop      r27
pop      r26
pop      r16
reti
```

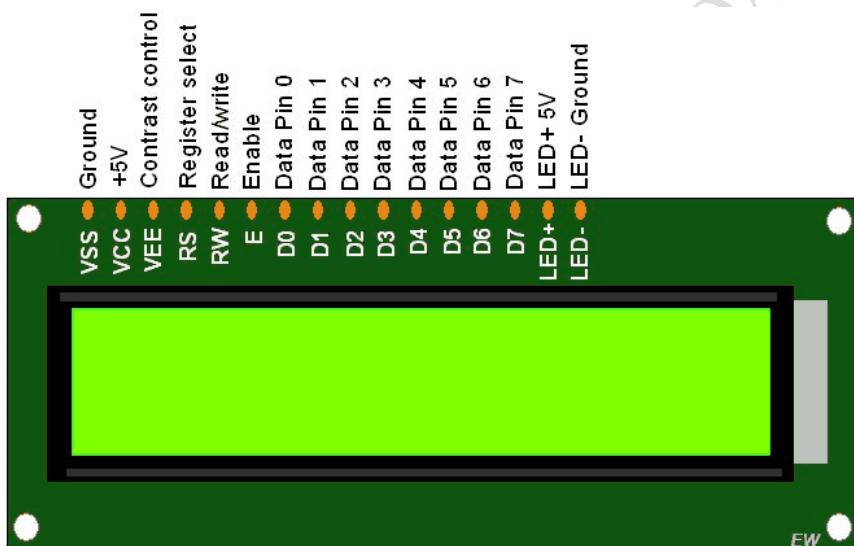
BỘ MÔN ĐIỆN TỬ - ĐH BÁCH KHOA TP.HCM

CHƯƠNG 6 HIỂN THỊ DÙNG LCD KÝ TỰ

6.1 LÝ THUYẾT CƠ BẢN

Để có thể hiển thị thông tin linh hoạt và tiết kiệm năng lượng, hệ thống có thể sử dụng module LCD. Có nhiều loại module LCD, trong đó thông dụng là loại hiển thị 2 hàng 16 ký tự. Module LCD có thể được dùng để hiển thị các thông tin dạng ký tự. Vì được tích hợp sẵn bộ lái LCD nên việc điều khiển module LCD tương đối đơn giản.

Module LCD đã được thiết kế chuẩn để cho phép ta có thể giao tiếp với LCD do một hằng bát kỳ sản xuất với điều kiện là các LCD có sử dụng cùng IC điều khiển HD44780. Phần lớn các module LCD sử dụng giao tiếp 14 chân trong đó có 8 đường dữ liệu, 3 đường điều khiển và 3 đường cấp nguồn. Kết nối được bố trí dưới dạng 1 hàng 14 chân hoặc 2 hàng 7 chân.

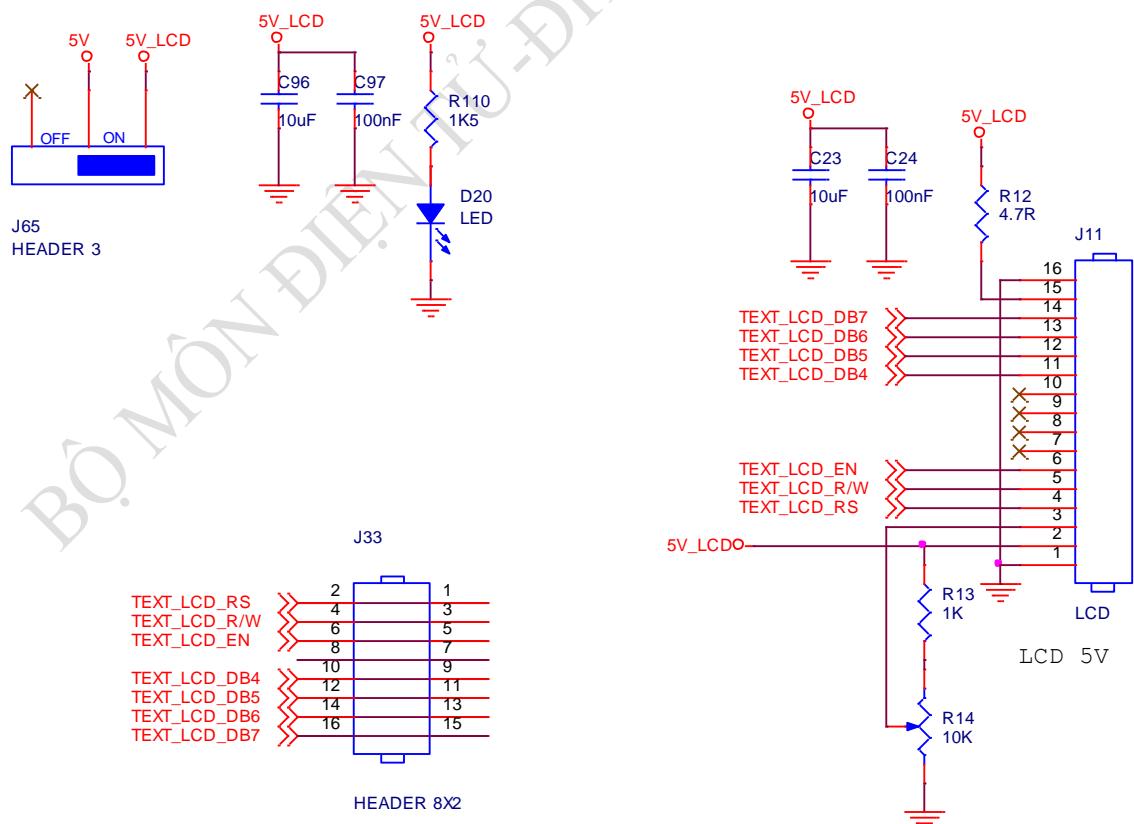


Các chân 1 và 2 là các chân cấp nguồn Vss, Vdd. Chân 3 Vee là chân điều khiển độ tương phản của màn hình. Chân 4 là đường RS, đây là chân điều khiển lệnh. Khi RS = 0 thì dữ liệu ghi vào LCD được hiểu là các lệnh, dữ liệu đọc từ LCD được hiểu là trạng thái của nó. Chân 5 là đường điều khiển đọc ghi R/nW, mức thấp sẽ cho phép ghi vào LCD, mức cao cho phép đọc ra từ LCD. Chân 6 là đường điều khiển cho phép E. Các chân còn lại chứa dữ liệu 8-bit vào hoặc ra LCD.

Chân số	Tên	Chức năng
1	V _{ss}	Đất
2	V _{DD}	Cực + của nguồn điện
3	V _{EE}	Tương phản (contrast)

4	RS	Register Select (Chọn thanh ghi)
5	R/W	Read/Write
6	E	Cho phép (Enable)
7	D0	Bit 0 của dữ liệu
8	D1	Bit 1 của dữ liệu
9	D2	Bit 2 của dữ liệu
10	D3	Bit 3 của dữ liệu
11	D4	Bit 4 của dữ liệu
12	D5	Bit 5 của dữ liệu
13	D6	Bit 6 của dữ liệu
14	D7	Bit 7 của dữ liệu

6.2 THIẾT KẾ PHẦN CỨNG



Hình 21 Sơ đồ giao tiếp LCD

LCD được thiết kế để với giao tiếp song song, với độ rộng bit là 4. Để điều khiển LCD, ta kết nối header J33 đến 1 port của AVR, sử dụng bus dây hoặc các dây đơn và cấp nguồn bằng cách đưa jumper J65 sang vị trí ON.

6.3 LẬP TRÌNH GIAO TIẾP LCD

Module LCD được điều khiển thông qua một tập lệnh chuẩn. Bảng sau tóm tắt các lệnh điều khiển LCD.

Lệnh	Chức năng
0F	LCD bật, con trỏ bật, con trỏ nhấp nháy bật
01	Xoá toàn màn hình
02	Quay về màn hình chính
04	Giảm con trỏ
06	Tăng con trỏ
0E	Màn hình bật, con trỏ nhấp nháy tắt
80	Bắt con trỏ trở về vị trí đầu tiên của hàng 1
C0	Bắt con trỏ trở về vị trí đầu tiên của hàng 2
38	Sử dụng 2 hàng và ma trận 5x7
83	Con trỏ hàng 1 vị trí 3
3C	Kích hoạt dòng 2
08	Tắt màn hình hiển thị và con trỏ
C1	Nhảy đến dòng 2 vị trí 1
0C	Bật màn hình hiển thị, tắt con trỏ
C2	Nhảy đến hàng 2, vị trí 2
0C	Display ON, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2

Bảng 4 Các lệnh của LCD

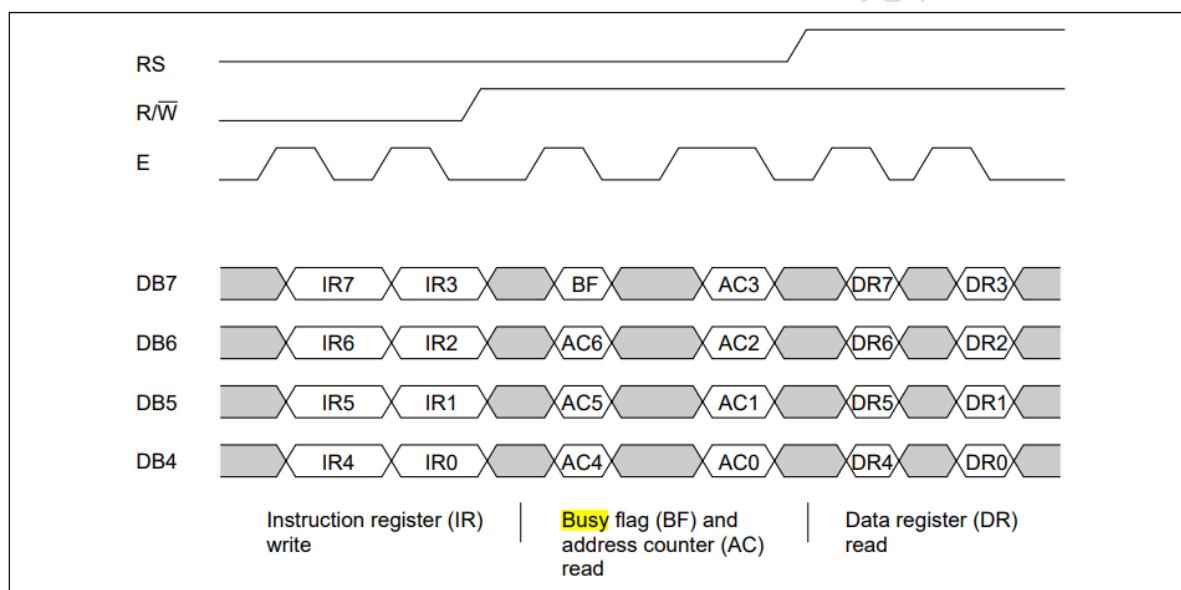
Để gửi một lệnh ra LCD, ta đặt mã lệnh lên các tín hiệu DATA, tín hiệu RS = 0 để chọn thanh ghi lệnh, tín hiệu RW bằng 0 để chọn chế độ ghi, tín hiệu EN ban đầu bằng 0. Sau đó, ta tạo 1 xung trên EN bằng cách đưa tín hiệu EN lên 1 và xuống 0 để ghi vào thanh ghi lệnh.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Để ghi dữ liệu ra LCD để hiển thị, ta đặt dữ liệu lên các tín hiệu DATA, tín hiệu RS = 1 để chọn thanh ghi data, tín hiệu RW bằng 0 để chọn chế độ ghi, tín hiệu EN ban đầu bằng 0. Sau đó, ta tạo 1 xung trên EN bằng cách đưa tín hiệu EN lên 1 và xuống 0 để ghi vào thanh ghi data.

Để biết được LCD có sẵn sàng để nhận lệnh hay không, ta có thể đọc thanh ghi lệnh. Tín hiệu RS = 0 để chọn thanh ghi lệnh, tín hiệu RW bằng 1 để chọn chế độ đọc, tín hiệu EN ban đầu bằng 0. Sau đó, ta tạo 1 xung trên EN bằng cách đưa tín hiệu EN lên 1 và xuống 0. Khi đó dữ liệu của LCD sẽ xuất ra trên các chân dữ liệu.

Ở chế độ 4 bit (được kết nối như trên kit thí nghiệm), dữ liệu mỗi lần truyền/nhận là 1 nibble 4 bit, với nibble cao được truyền/nhận trước. Vì vậy, mỗi quá trình ghi/đọc dữ liệu cần 2 xung EN. Giản đồ xung sau thể hiện quá trình ghi/đọc LCD.



Hình 22 Giản đồ xung điều khiển LCD

Đoạn mã sau thực hiện việc ghi lệnh vào LCD.

```
; Subroutine to send command to LCD
;Command code in r16
;LCD_D7..LCD_D4 connect to PA7..PA4
;LCD_RS connect to PA0
;LCD_RW connect to PA1
;LCD_EN connect to PA2
LCD_Send_Command:
    push  r17
    call   LCD_wait_busy ; check if LCD is busy
    mov    r17,r16          ;save the command
    ; Set RS low to select command register
    ; Set RW low to write to LCD
    andi   r17,0xF0
    ; Send command to LCD
    out   LCDPORT, r17
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
nop
nop
; Pulse enable pin
sbi LCDPORT, LCD_EN
nop
nop
nop
cbi LCDPORT, LCD_EN
swap    r16
andi   r16,0xF0
; Send command to LCD
out LCDPORT, r16
; Pulse enable pin
sbi LCDPORT, LCD_EN
nop
nop
cbi LCDPORT, LCD_EN
pop    r17
ret
```

Đoạn mã sau thực hiện việc ghi dữ liệu ra LCD

```
LCD_Send_Data:
push   r17
call   LCD_wait_busy ;check if LCD is busy
mov    r17,r16          ;save the command
; Set RS high to select data register
; Set RW low to write to LCD
andi   r17,0xF0
ori    r17,0x01
; Send data to LCD
out LCDPORT, r17
nop
; Pulse enable pin
sbi LCDPORT, LCD_EN
nop
cbi LCDPORT, LCD_EN
; Delay for command execution
;send the lower nibble
nop
swap   r16
andi   r16,0xF0
; Set RS high to select data register
; Set RW low to write to LCD
andi   r16,0xF0
ori    r16,0x01
; Send command to LCD
out LCDPORT, r16
nop
; Pulse enable pin
sbi LCDPORT, LCD_EN
nop
cbi LCDPORT, LCD_EN
pop    r17
ret
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Như thể hiện trên Hình 22, sau mỗi tác vụ ghi lệnh hay dũ liệu, LCD cần một thời gian để xử lý. Trong thời gian này, LCD sẽ không nhận các lệnh hay dũ liệu mới, vì vậy, khi gửi dũ liệu hay lệnh vào LCD ta phải đảm bảo là LCD không ở trong trạng thái busy.

Thời gian xử lý này của LCD phụ thuộc vào loại lệnh. Bảng sau thể hiện thời gian cần thiết cho một số lệnh tiêu biểu

Instruction	Code										Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μ s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μ s
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μ s

Hình 23 Thời gian trễ của LCD

Cách đơn giản nhất là sau mỗi lệnh ghi/đọc vào CPU, ta làm chương trình chờ khoảng 2 ms trước khi thực hiện lệnh kế tiếp. Tuy nhiên, cách làm này sẽ làm cho chương trình không đạt hiệu năng cao nhất nếu ta cho thời gian trễ quá dài. Nếu ta giảm thời gian trễ này thì có thể làm LCD hoạt động sai.

Cách thứ 2 là trước khi thực hiện lệnh, ta kiểm tra trạng thái của LCD bằng cách đọc thanh ghi lệnh và kiểm tra bit thứ 7 (BUSY FLAG). Bit này được xóa về 0 khi LCD sẵn sàng.

```
LCD_wait_busy:  
    push r16  
    ldi r16, 0b00000111 ; set PA7-PA4 as input, PA2-PA0 as output  
    out LCDPORTDIR, r16  
    ldi r16, 0b11110010 ; set RS=0, RW=1 for read the busy flag  
    out LCDPORT, r16  
    nop  
LCD_wait_busy_loop:  
    sbi LCDPORT, LCD_EN  
    nop  
    nop  
    in r16, LCDPORTPIN  
    cbi LCDPORT, LCD_EN  
    nop
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
sbi LCDPORT, LCD_EN
nop
nop
cbi LCDPORT, LCD_EN
nop
andi r16,0x80
cpi      r16,0x80
breq    LCD_wait_busy_loop
ldi r16, 0b11110111 ; set PA7-PA4 as output, PA2-PA0 as output
out LCDPORTDIR, r16
ldi r16,0b00000000      ; set RS=0, RW=1 for read the busy flag
out LCDPORT, r16
pop r16
ret
```

Khi mới bật nguồn, LCD cần phải chờ một khoảng thời gian khoảng 15 ms để hoạt động ổn định. Sau đó, ta có thể bắt đầu khởi động để đưa LCD vào chế độ hoạt động phù hợp bằng cách gửi các lệnh vào LCD. Chương trình con sau làm nhiệm vụ khởi động một LCD với kết nối như trong chú thích.

```
;init the LCD
;LCD_D7..LCD_D4 connect to PA7..PA4
;LCD_RS connect to PA0
;LCD_RW connect to PA1
;LCD_EN connect to PA2

.equ LCDPORT = PORTA ; Set signal port reg to PORTA
.equ LCDPORTDIR = DDRA ; Set signal port dir reg to PORTA
.equ LCDPORTPIN = PINA ; Set clear signal port pin reg to PORTA
.equ LCD_RS = PINA0
.equ LCD_RW = PINA1
.equ LCD_EN = PINA2
.equ LCD_D7 = PINA7
.equ LCD_D6 = PINA6
.equ LCD_D5 = PINA5
.equ LCD_D4 = PINA4

LCD_Init:
; Set up data direction register for Port A
ldi r16, 0b11110111 ; set PA7-PA4 as outputs, PA2-PA0 as output
out LCDPORTDIR, r16
; Wait for LCD to power up
call DELAY_10MS
call DELAY_10MS

; Send initialization sequence
ldi r16, 0x02 ; Function Set: 4-bit interface
call LCD_Send_Command
ldi r16, 0x28 ; Function Set: enable 5x7 mode for chars
call LCD_Send_Command
ldi r16, 0x0E ; Display Control: Display OFF, Cursor ON
call LCD_Send_Command
ldi r16, 0x01 ; Clear Display
call LCD_Send_Command
ldi r16, 0x80 ; Clear Display
call LCD_Send_Command
ret
```

6.4 VIẾT CHƯƠNG TRÌNH HIỂN THỊ KÝ TỰ LÊN LCD

Để hiển thị ký tự lên LCD, ta di chuyển con trỏ đến vị trí cần thiết và ghi mã ASCII của ký tự muốn hiển thị vào LCD.

Ví dụ sau là chương trình chính khởi động LCD và xuất một số ký tự ra LCD

```
.include "m324padef.inc" ; Include Atmega324pa definitions
.org 0x0000 ; interrupt vector table
rjmp reset_handler ; reset
;***** Program ID *****
.org    INT_VECTORS_SIZE
course_name:
.db      "TN Vi Xu Ly-AVR",0
course_time:
.db      "HK2 2022-2023",0

reset_handler:
    call  LCD_Init
; display the first line of information
    ldi   ZH, high(course_name)      ; point to the information that is to be
displayed
    ldi   ZL, low(course_name)
    call  LCD_Send_String
    ldi   r16,1
    ldi   r17,0
    call  LCD_Move_Cursor
    ldi   ZH, high(course_time)      ; point to the information that is to be
displayed
    ldi   ZL, low(course_time)
    call  LCD_Send_String
start:
    rjmp start
```

Ví dụ 15 Hiển thị ký tự ra LCD

Trong ví dụ trên sử dụng hàm LCD Move Cursor, với mã nguồn như dưới đây

```
; Function to move the cursor to a specific position on the LCD
; Assumes that the LCD is already initialized
; Input: Row number in R16 (0-based), Column number in R17 (0-based)
LCD_Move_Cursor:
    cpi   r16,0 ;check if first row
    brne LCD_Move_Cursor_Second
    andi  r17, 0x0F
    ori   r17, 0x80
    mov   r16,r17
; Send command to LCD
    call LCD_Send_Command
    ret
LCD_Move_Cursor_Second:
    cpi   r16,1 ;check if second row
    brne LCD_Move_Cursor_Exit      ;else exit
    andi  r17, 0x0F
    ori   r17, 0xC0
    mov   r16,r17
; Send command to LCD
    call LCD_Send_Command
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
LCD_Move_Cursor_Exit:  
    ; Return from function  
    ret
```

Ví dụ 16 Hàm di chuyển con trỏ trên LCD

Để gửi một chuỗi ra LCD, ta có thể định nghĩa chuỗi trong ROM và gửi tuần tự các ký tự ra LCD cho đến khi kết thúc (gặp ký tự 0). Đoạn chương trình sau thực hiện việc này

```
; Subroutine to send string to LCD  
; address of the string on ZH-ZL  
; string end with Null  
.def LCDData = r16  
LCD_Send_String:  
    push    ZH                      ; preserve pointer registers  
    push    ZL  
    push    LCDData  
  
    ; fix up the pointers for use with the 'lpm' instruction  
    lsl     ZL                      ; shift the pointer one bit left for the  
lpm instruction  
    rol     ZH  
    ; write the string of characters  
LCD_Send_String_01:  
    lpm    LCDData, Z+              ; get a character  
    cpi    LCDData, 0               ; check for end of string  
    breq   LCD_Send_String_02      ; done  
  
    ; arrive here if this is a valid character  
    call   LCD_Send_Data          ; display the character  
    rjmp   LCD_Send_String_01      ; not done, send another character  
  
    ; arrive here when all characters in the message have been sent to the LCD module  
LCD_Send_String_02:  
    pop    LCDData  
    pop    ZL                      ; restore pointer registers  
    pop    ZH  
    ret
```

BỘ MÔN ĐIỀU KIỂM VÀ THỰC HIỆN TINH TẾ
HỌC KÌ 1 - KHÓA HỌC 2018-2019

CHƯƠNG 7 THÍ NGHIỆM GIAO TIẾP QUA CỔNG NỐI TIẾP

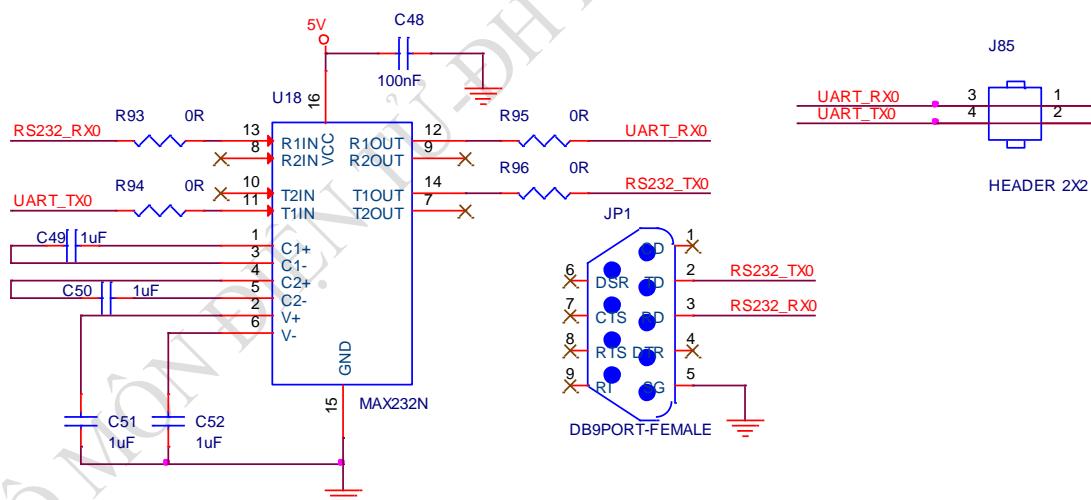
7.1 LÝ THUYẾT CƠ BẢN

Truyền thông nối tiếp là một phần rất quan trọng trong các hệ thống xử lý. Đối với các vi điều khiển, việc truyền nối tiếp được thực hiện thông qua ngoại vi cổng nối tiếp. Ngoại vi này cho phép người lập trình giao tiếp với bên ngoài thông qua giao thức truyền nối tiếp 8 bit dữ liệu, 1 stop bit. Tốc độ truyền có thể lập trình được bằng phần mềm.

Để truyền dữ liệu đi xa, mức điện áp TTL từ vi điều khiển sẽ được chuyển sang các mức RS232 hoặc 485 bằng các vi mạch chuyển mức. Trên kit thí nghiệm, cổng nối tiếp của AVR có thể giao tiếp với cổng RS-232 trên máy tính thông qua một vi mạch MAX232 chuyển đổi từ TTL sang RS. Việc truyền thông tin chỉ cần 3 dây TXD, RXD, và GND nếu không dùng bắt tay bằng phần cứng.

7.2 THIẾT KẾ PHẦN CỨNG

Trên kit thí nghiệm, cổng nối tiếp của AVR có thể giao tiếp với cổng RS-232 trên máy tính thông qua một vi mạch MAX232 chuyển đổi từ TTL sang RS. Việc truyền thông tin chỉ cần 3 dây TXD, RXD, và GND nếu không dùng bắt tay bằng phần cứng.



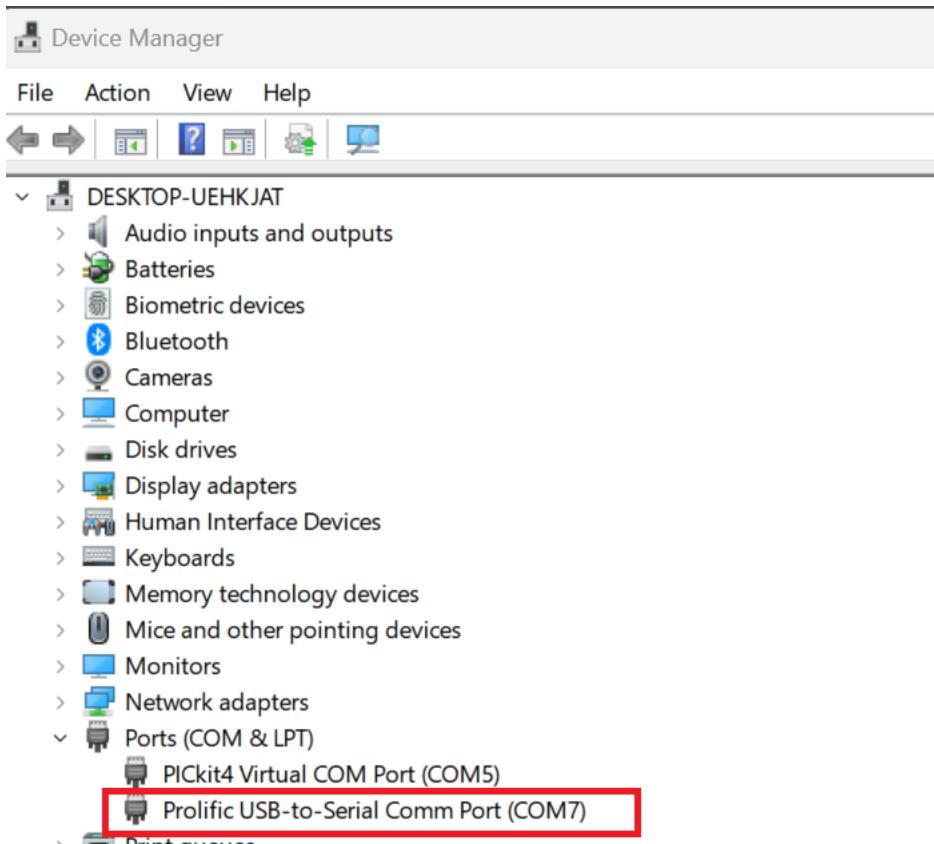
Hình 24 Thiết kế khối giao tiếp UART-RS232

7.3 KẾT NỐI PHẦN CỨNG VÀ CẤU HÌNH PHẦN MỀM

Để bắt đầu lập trình, sinh viên kết nối tín hiệu TX và RX trên header J85 của khối RS232 vào chân PD1 (TXD0) và chân PD0 (RXD0) của vi điều khiển. Kết nối dây cáp chuyển đổi USB-RS232 vào cổng COM trên kit thí nghiệm. Kết nối connector USB của cáp này vào cổng USB trên máy tính. Sử dụng phần mềm Putty để giao tiếp với cổng COM.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

1. Mở Device Manager trên máy tính và ghi nhận số cổng COM ảo mà máy tính nhận ra.
Đây sẽ là cổng COM để giao tiếp với UART0 của vi điều khiển.

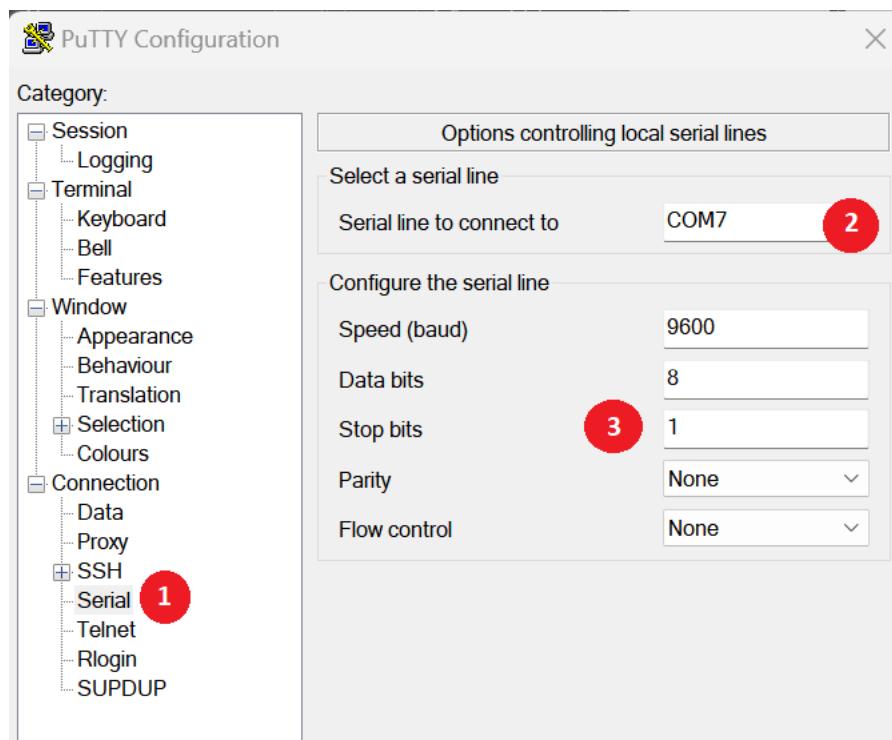


Hình 25 Ghi nhận cổng COM ảo trên máy tính

2. Mở Putty. Chọn tab Serial, nhập vào trường Serial Line cổng COM chính xác, sau đó cấu hình baudrate, số data bit, số parity bit, số stop bit. Chọn Flow control là None

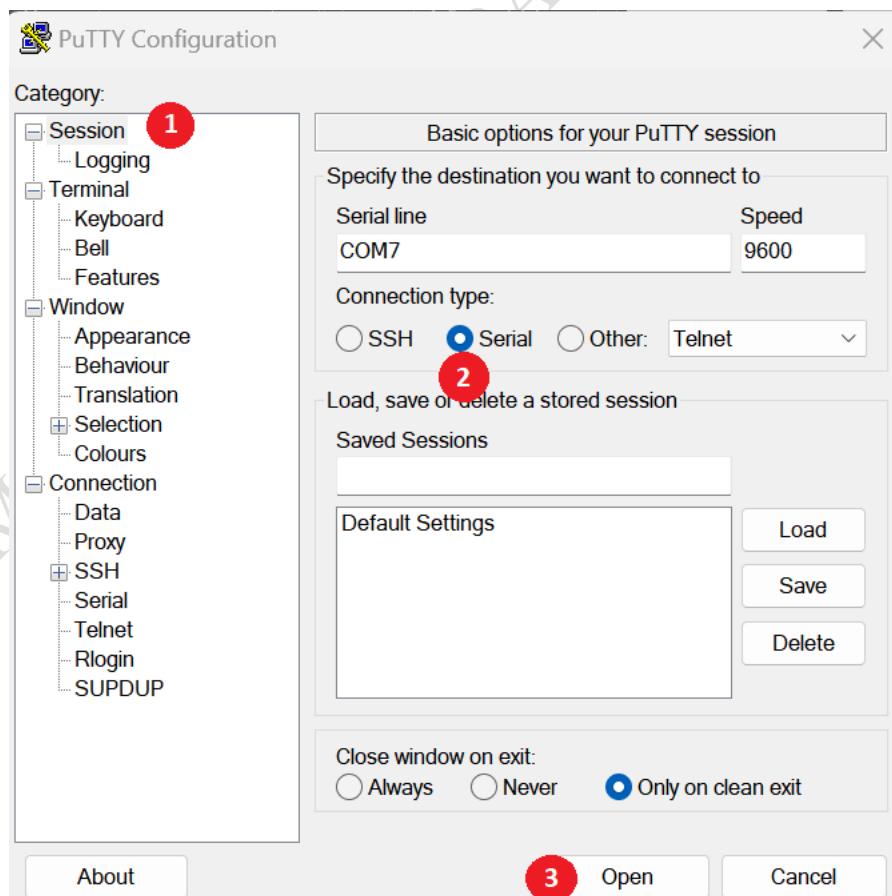
BỘ MÔN ĐIỀU HÒA

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ



Hình 26 Cấu hình cổng COM

- Chọn tab section, click chọn Serial và Click Open để mở Console



Hình 27 Mở Console

7.4 LẬP TRÌNH CHO ATMEGA324 GIAO TIẾP UART

Tùy theo cấu hình cầu chì trên vi điều khiển mà ta sẽ có tần số xung nhịp khác nhau để tính toán baudrate. Trên kit mặc định sử dụng thạch anh ngoài với tần số 8Mhz. Ta có thể tham khảo các giá trị để tính toán baudrate và sai số tương ứng trong datasheet của Atmega324.

Hình 28 Bảng giá trị UBRRn để cấu hình baudrateHình 28 mô tả các giá trị cần nạp vào thanh ghi UBRR ứng với các tần số CPU 1Mhz, 1.8432 Mhz, 2Mhz. Với tần số 1Mhz, và baudrate là 9600, để hạn chế sai số ta chọn chế độ double speed, và cấu hình thanh ghi UBRR với giá trị là 12.

Baud Rate [bps]	$f_{osc} = 1.0000MHz$				$f_{osc} = 1.8432MHz$				$f_{osc} = 2.0000MHz$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	-	-	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	-	-	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	-	-	-	-	-	-	0	0.0%	-	-	-	-
250k	-	-	-	-	-	-	-	-	-	-	0	0.0%
Max.(1)	62.5kbps		125kbps		115.2kbps		230.4kbps		125kbps		250kbps	

Hình 28 Bảng giá trị UBRRn để cấu hình baudrate

Ví dụ 17 mô tả quá trình cấu hình UART0 ở baudrate 9600, và thực hiện việc chờ nhận một byte ở UART0, sau đó truyền ngược lại UART0.

```
.include "m324padef.inc"
; Replace with your application code
    call    USART_Init
start:
    call    USART_ReceiveChar
    call    USART_SendChar
    rjmp start
;init UART 0
;CPU clock is 1Mhz
USART_Init:
    ; Set baud rate to 9600 bps with 1 MHz clock
    ldi r16, 12
    sts UBRR0L, r16
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
;set double speed
ldi r16, (1 << U2X0)
sts UCSR0A, r16
; Set frame format: 8 data bits, no parity, 1 stop bit
ldi r16, (1 << UCSZ01) | (1 << UCSZ00)
sts UCSR0C, r16
; Enable transmitter and receiver
ldi r16, (1 << RXEN0) | (1 << TXEN0)
sts UCSR0B, r16
ret
;send out 1 byte in r16
USART_SendChar:
    push r17
    ; Wait for the transmitter to be ready
    USART_SendChar_Wait:
        lds r17, UCSR0A
        sbrs r17, UDRE0      ;check USART Data Register Empty bit
        rjmp USART_SendChar_Wait
        sts UDR0, r16        ;send out
        pop r17
    ret
;receive 1 byte in r16
USART_ReceiveChar:
    push r17
    ; Wait for the transmitter to be ready
    USART_ReceiveChar_Wait:
        lds r17, UCSR0A
        sbrs r17, RXC0      ;check USART Receive Complete bit
        rjmp USART_ReceiveChar_Wait
        lds r16, UDR0        ;get data
        pop r17
    ret
```

Ví dụ 17 Cấu hình và giao tiếp UART

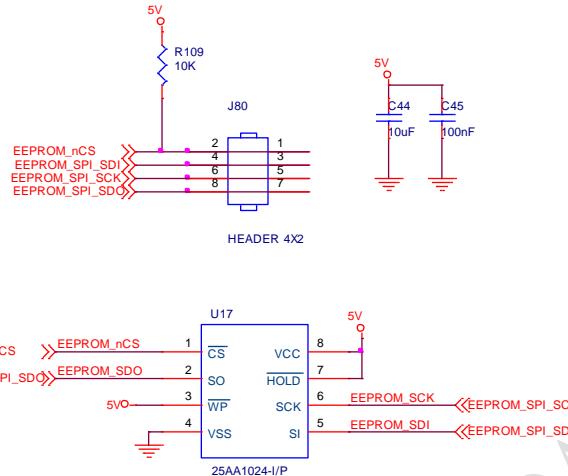
BỘ MÔN ĐIỀU TỰ ĐỘNG HÓA
ĐH BÁCH KHOA TP.HCM

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

BỘ MÔN ĐIỆN TỬ - ĐH BÁCH KHOA TP.HCM

CHƯƠNG 8 BỘ NHỚ EEPROM VÀ SDCARD

8.1 THIẾT KẾ PHẦN CỨNG KHỐI EEPROM



Hình 29 Thiết kế phần cứng khối EEPROM

Chip nhớ 25AA1024 là một chip EEPROM có dung lượng lên đến 1Mbit, giao tiếp theo chuẩn SPI. Trong các thiết kế cần lưu giữ các thông tin mà không bị xóa khi mất điện, ví dụ như thông tin cấu hình, password,..., ta thường sử dụng các chip nhớ dạng này.

Name	Pin Number	Function
CS	1	Chip Select Input
SO	2	Serial Data Output
WP	3	Write-Protect Pin
Vss	4	Ground
SI	5	Serial Data Input
SCK	6	Serial Clock Input
HOLD	7	Hold Input
VCC	8	Supply Voltage

Bảng 5 Mô tả chân tín hiệu của EEPROM

Các tín hiệu điều khiển của chip nhớ được kết nối ra header J80. Ta có thể kết nối các tín hiệu này đến các chân port của vi điều khiển để thực hiện ghi/đọc vào chip.

Trên thiết kế, các tín hiệu /WP và /HOLD được nối lên VCC, do đó các tín hiệu này không tích cực.

8.2 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

Atmega32 có phần cứng SPI onchip, có thể hoạt động ở cả chế độ master và slave. Chip EEPROM trên hoạt động như là một thiết bị SPI Slave. Để giao tiếp với nó, ta sẽ cấu hình vi điều khiển AVR ở chế độ master.

Kết nối các chân của AVR và tín hiệu điều khiển trên header J80. Các tín hiệu SPI gồm có MOSI, MISO, SCK và SS.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

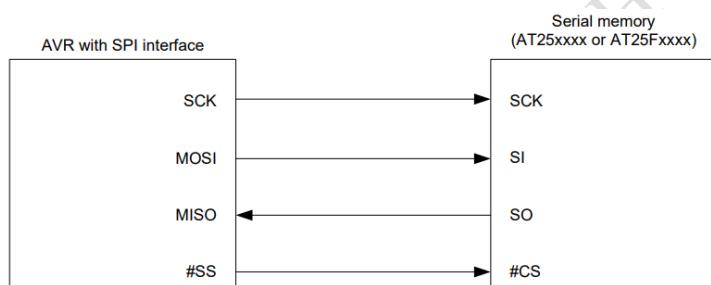
Trên ATMEGA324, các chân chức năng SPI được kết nối vào các chân port như sau:

32-pin TQFP/ QFN/ MLF Pin #	40-pin PDIP Pin #	DRQFN Pin#	VFBGA Pin#	PAD	EXTINT	PCINT	ADC/AC	OSC	T/C # 0	T/C # 1	USART	I2C	SPI
1	6	A1	B2	PB[5]		PCINT13							MOSI
2	7	B1	B1	PB[6]		PCINT14							MISO
3	8	A2	C3	PB[7]		PCINT15							SCK

Đối với tín hiệu Slave Select (SS), nếu làm việc ở chế độ SPI Slave, trên ATMEGA324 kết nối vào chân PB4. Ở chế độ master, ta có thể sử dụng bất kỳ một chân port nào và điều khiển bằng phần mềm.

Nếu kết nối nhiều SPI Slave chip vào cùng 1 bus SPI, các slave này sẽ sử dụng chung các tín hiệu MOSI, MISO, SCK. Các tín hiệu SS của các slave sẽ kết nối vào các chân I/O port của AVR.

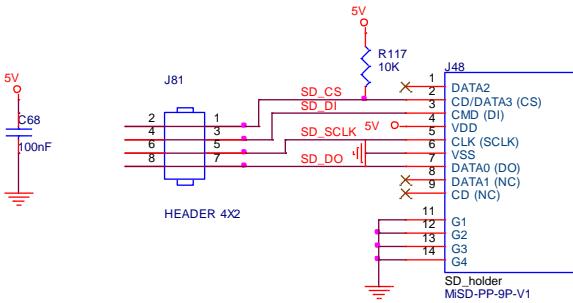
Các tín hiệu của EEPROM được kết nối với các tín hiệu SPI như sau:



Hình 30 Kết nối tín hiệu SPI giữa AVR và EEPROM

Chi tiết các lệnh ghi/đọc và giản đồ xung, sinh viên tham khảo chi tiết trong datasheet của 25AA1024.

8.3 THIẾT KẾ PHẦN CỨNG KHỐI SDCARD



Hình 31 Thiết kế phần cứng khống EEPROM

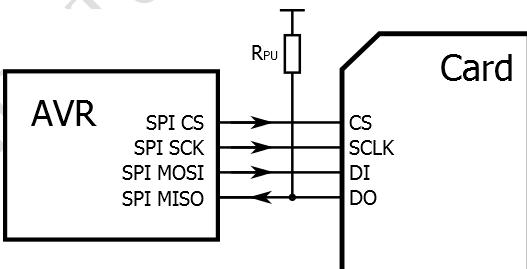
Để lưu trữ các dữ liệu có kích thước lớn, như hình ảnh, âm thanh, dữ liệu trong thời gian dài, ta thường thiết kế sử dụng SDCARD.

Các SDCARD có 2 giao diện điều khiển: SDIO hoặc SPI. Với giao diện SDIO, dữ liệu được truyền nhận mỗi lần 4 bit, nên có tốc độ cao. Giao diện SPI dữ liệu được truyền nối tiếp, nên có tốc độ chậm hơn. Tuy nhiên, phần lớn các vi điều khiển chỉ hỗ trợ SPI mà không có SDIO, nên ta thường sử dụng SPI để giao tiếp SDCARD.

8.4 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

Tương tự kết nối với SPI EEPROM, Các tín hiệu của SDCARD được kết nối với các tín hiệu SPI như sau.

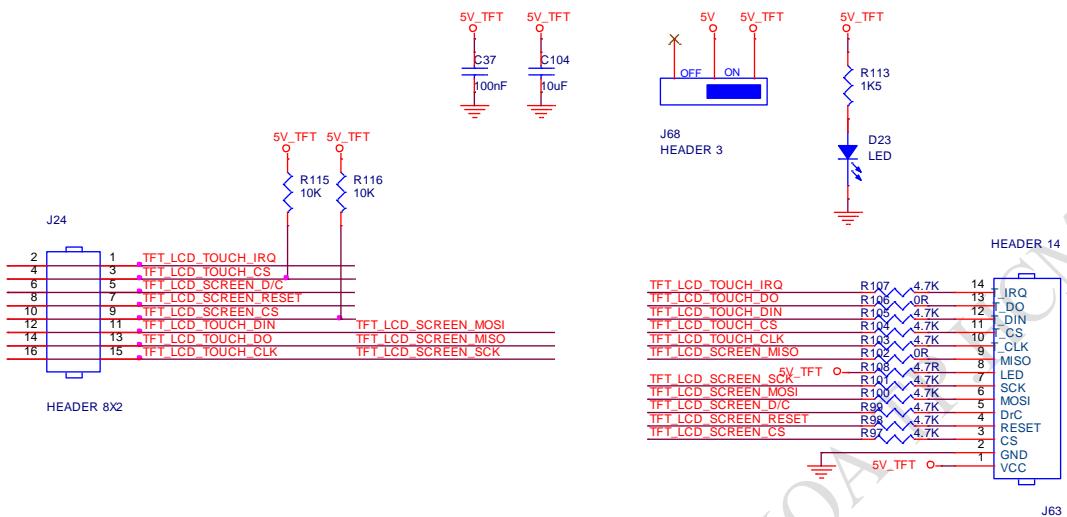
Trên thiết kế không có điện trở kéo lên bên ngoài cho chân DO của SDCARD. Ta phải kích hoạt điện trở kéo lên bên trong của chân MISO của AVR.



Hình 32 Kết nối tín hiệu SPI giữa AVR và SDCARD

CHƯƠNG 9 TOUCH SCREEN LCD

9.1 THIẾT KẾ PHẦN CỨNG KHỐI LCD



Hình 33 Thiết kế phần cứng khối LCD Graphic

Để hiển thị các giao diện graphic và sử dụng màn hình cảm ứng, trên kit thí nghiệm được tích hợp 1 LCD TFT 3.2 inch. Màn hình LCD này có chip điều khiển ILI9341, và chip XPT2046 điều khiển màn cảm ứng.

LCD và touch screen đều giao tiếp qua SPI. Header J63 trên **Error! Reference source not found.** là connector để kết nối các tín hiệu trên module LCD touch.

Tín hiệu từ module LCD được kết nối về header J24, từ đó có thể kết nối về vi điều khiển.

Các tín hiệu SCK, MOSI, MISO của chip ILI9341 và XPT2046 được nối chung tại header J24.

Các tín hiệu chip select của 2 chip này được điều khiển riêng.

Ngoài các tín hiệu kể trên, LCD còn có tín hiệu chọn lệnh/dữ liệu (D/C) và tín hiệu reset. Với màn cảm ứng, có tín hiệu TOUCH_IRQ tích cực khi có sự kiện chạm vào màn hình. Các tín hiệu này được kết nối vào J24

9.2 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

Ta sẽ kết nối các tín hiệu MOSI, MISO và SCK của AVR vào các tín hiệu tương ứng trên J24.

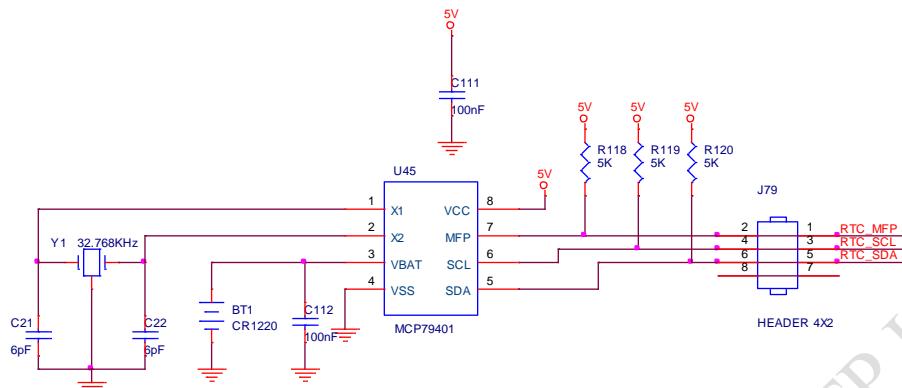
Các tín hiệu chip select, LCD_D/C, LCD_RESET được kết nối vào các chân port. Tín hiệu TOUCH_IRQ được kết nối vào một chân ngắt ngoài.

Cách tương tác với LCD và touch screen, sinh viên tham khảo tài liệu:

http://www.lcdwiki.com/3.2inch_SPI_Module_ILI9341_SKU:MSP3218

CHƯƠNG 10 ĐỒNG HỒ THỜI GIAN THỰC (RTC)

10.1 THIẾT KẾ PHẦN CỨNG KHỐI RTC



Hình 34 Thiết kế phần cứng khối RTC

Chip MCP79401 là vi mạch thời gian thực của Microchip, có tính năng đồng hồ thời gian thực/đ lịch, tự động cập nhật ngày tháng năm, giờ phút giây, năm nhuận. Trong các thiết kế cần thời gian thực, ta thường sử dụng các chip này thay vì sử dụng CPU để lập trình đếm thời gian. MCP79401 là một thiết bị I2C Slave, giao tiếp với vi điều khiển AVR qua bus I2C. Ngoài ra, tín hiệu MFP có thể lập trình để báo hiệu cho vi điều khiển một sự kiện.

Các tín hiệu điều khiển của RTC được kết nối ra header J79.

10.2 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

TWI (Two-Wire Serial Interface) là một module truyền thông nối tiếp đồng bộ trên các chip AVR dựa trên chuẩn truyền thông I2C. Trên ATMEGA324, các chân chức năng TWI được kết nối vào các chân port như sau:

PC[0]		PCINT16				SCL		
PC[1]		PCINT17				SDA		

Để giao tiếp với IC MCP79401, ta kết nối chân PC0 vào tín hiệu RTC_SCL, chân PC1 vào RTC_SDA.

Ta có thể kết nối chân RTC_MFP vào một chân ngắt ngoài hoặc một chân port có hỗ trợ ngắt PINCHANGE. Khi đó, ta lập trình chân này để kích hoạt AVR khi có sự kiện xảy ra.

Trong trường hợp ta muốn lập trình cho AVR cập nhật liên tục thời gian từ RTC (VD: ứng dụng đồng hồ), ta có thể lập trình cho chân MFP xuất ra 1 xung tần số 1Hz, và dùng tín hiệu này để kích khởi 1 ngắt. Khi đó, cứ mỗi lần ngắt kích khởi ta sẽ cập nhật dữ liệu từ RTC mà không cần phải hỏi vòng.

Chi tiết về các thanh ghi, địa chỉ của chip RTC, sinh viên tham khảo datasheet của MCP79401.

CHƯƠNG 11 GIAO TIẾP ADC VÀ KHỐI CẢM BIẾN TƯƠNG TỰ

11.1 THIẾT KẾ GIAO TIẾP ADC

11.1.1 TỔNG QUAN VỀ ADC TRÊN ATMEGA324PA

ATMEGA324PA có 1 bộ ADC 10 bit, với 8 kênh ngõ vào đơn cực ADC7..ADC0. Các kênh đơn cực này có thể được sử dụng thành 16 cặp ngõ vào vi sai.

Sau mỗi lần chuyển đổi ADC, kết quả được lưu trong cặp thanh ghi ADCH-ADCL.

Với chuyển đổi ADC đơn cực, kết quả đọc ADC sẽ là:

$$ADCVal = \frac{V_{IN} * 1024}{V_{REF}} \quad (1)$$

Với chuyển đổi ADC vi sai, kết quả đọc ADC sẽ là:

$$ADCVal = \frac{(V_{POS} - V_{NEG}) * GAIN * 512}{V_{REF}} \quad (2)$$

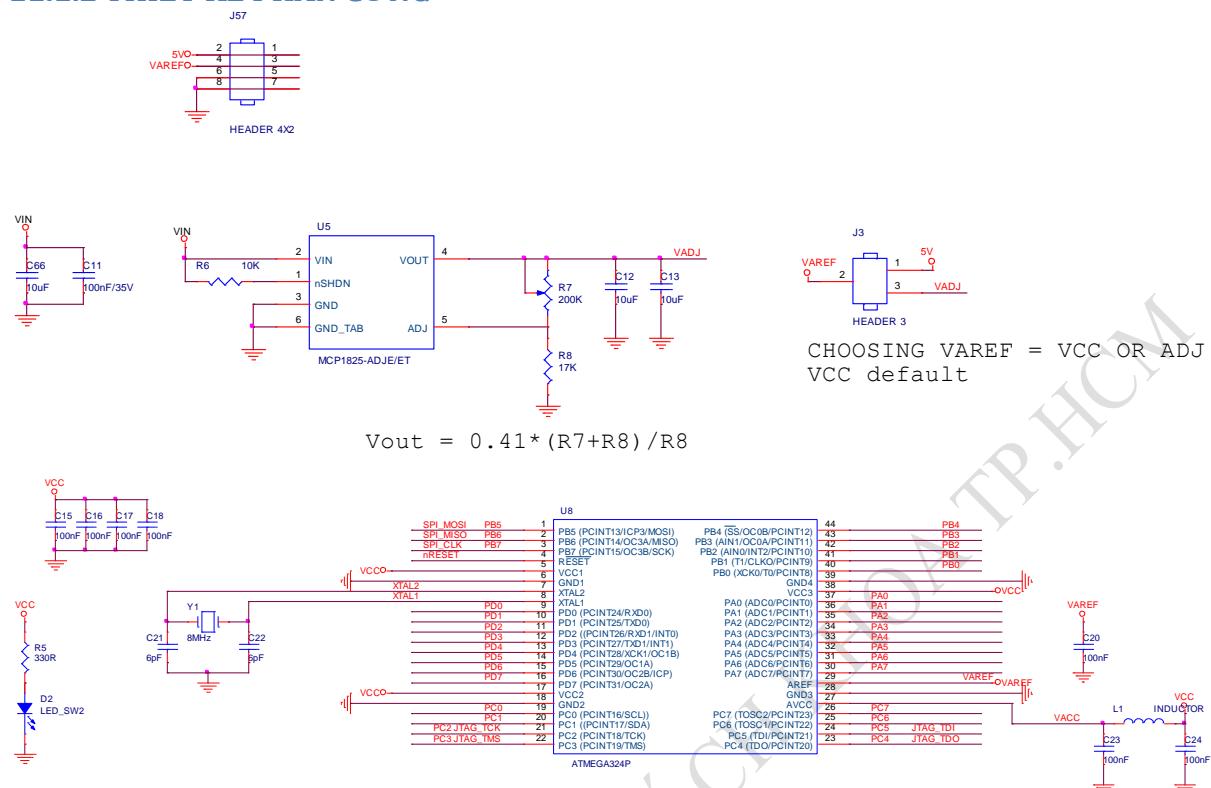
Với ADC vi sai, kết quả ở dạng bù 2, từ 0x200 (-512) đến 0x1FF (+511).

Điện áp tham chiếu V_{REF} có thể được đưa vào từ bên ngoài bằng cách cấp 1 điện áp lên chân AV_{REF} , hoặc có thể chọn từ điện áp AVCC hoặc điện áp tham chiếu 2.56V bên trong. Lưu ý rằng nếu ta đã kết nối điện áp bên ngoài vào chân AV_{REF} , ta không được chọn điện áp AVCC hoặc điện áp tham chiếu bên trong, vì khi đó sẽ làm ngắn mạch hai nguồn bên ngoài và bên trong, dẫn đến hư hỏng chip.

Một lưu ý khác là ở chế độ ADC vi sai, điện áp tham chiếu AV_{REF} nhỏ nhất là 2V và lớn nhất là $AV_{CC} - 0.5V$.

Sinh viên tham khảo datasheet của Atmega324pa, chương ADC - Analog to Digital Converter và chương Electrical Characteristics, phần ADC characteristics để hiểu rõ các tính năng và tham số của ADC.

11.1.2 THIẾT KẾ PHẦN CỨNG



Hình 35 Thiết kế khối CPU và nguồn điện áp tham chiếu cho ADC

Vì điều khiển ATMEGA324 được cấp nguồn 5V vào các chân V_{CC}. Nguồn 5V được lọc qua mạch lọc hình PI và cấp vào chân AV_{CC}. Như vậy, AV_{CC} cũng được cấp nguồn 5V.

IC nguồn MCP1825-ADJ được sử dụng để tạo nguồn thay đổi theo công thức

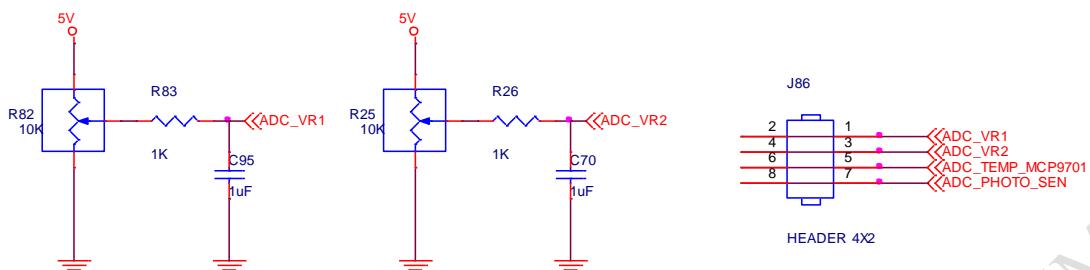
$$V_{OUT} = \frac{(R_7 + R_8) * 0.41}{R_8}$$

Header J3 dùng để chọn điện áp cho tham chiếu cho ADC. Tùy theo vị trí cáu hình jumper mà điện áp tham chiếu đưa vào sẽ là điện áp V_{CC} (5V) hoặc điện áp từ nguồn thay đổi. Trước khi chọn điện áp từ nguồn thay đổi, ta phải đảm bảo điện áp này không vượt quá 5V trong trường hợp dùng ADC đơn cực hoặc 4.5V trong trường hợp dùng ADC vi sai.

Nếu sử dụng điện áp tham chiếu nội (2.56V) hoặc AV_{CC}, ta phải gỡ bỏ jumper này để đảm bảo không có điện áp ngoài đưa vào chân AV_{REF}, nếu không sẽ làm hư hỏng CPU.

11.2 BIẾN TRỞ TẠO ÁP THAY ĐỔI

11.2.1 THIẾT KẾ PHẦN CỨNG



Hình 36 Sơ đồ khối biến trở

Trên kit thí nghiệm có 2 biến trở VR1 và VR2 dùng để tạo điện áp thay đổi. Các điện áp ra được kết nối vào header J86.

11.2.2 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

11.2.2.1 LẬP TRÌNH ADC Ở CHẾ ĐỘ ĐƠN CỰC

Chế độ ADC đơn cực được sử dụng để đo điện thế giữa một tín hiệu và GND. Để thử nghiệm, ta đưa điện áp từ các cầu phân áp tạo ra bởi VR1, VR2 vào các chân ngõ vào của ADC để lập trình. Kết nối tín hiệu ADC_VR1 hay ADC_VR2 vào chân port ADC7..ADC0 của AVR bằng các dây đơn.

Để đo giá trị điện áp, ta đọc giá trị trên ADC và tính ra điện áp đưa vào theo công thức:

$$V_{IN} = \frac{ADCVal * V_{REF}}{1024} \quad (3)$$

Trong đó ADCVal là giá trị đọc từ ADC, có giá trị không dấu từ 0x00 đến 0x3FF với ADC ở chế độ 10 bit.

Giá trị $\frac{V_{REF}}{1024}$ là giá trị của một LSB, là khoảng thay đổi điện áp ngõ vào làm cho giá trị ADC tăng lên 1 bit. Đây chính là độ phân giải của phép đo. Ví dụ với $V_{REF} = 5V$, độ phân giải của ADC 10 bit sẽ là: $1 \text{ LSB} = \frac{V_{REF}}{1024} = 0.0048828125 \text{ (V)}$.

Trong các phép đo, khi hiển thị ta thường mong muốn hiển thị với các độ phân giải 1, 0.1, 0.01...hoặc là bội số của các số trên, ví dụ 0.2, 0.02.. Khi đó, các điện áp tham chiếu được chọn là một lũy thừa của 2, ví dụ 1.024V, 2.56V, 2.048V, 4.096V.

Ví dụ ta chọn điện áp tham chiếu là 2.56V cho phép đo điện áp ở trên, khi đó độ phân giải của phép đo sẽ là: $1LSB = \frac{2.56}{1024} = 0.0025 \text{ V}$. Giá trị này thuận lợi để tính toán và hiển thị.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Để tính toán sử dụng lệnh assembly, ta cần dùng các thư viện các lệnh nhân 8 bit, 16 bit. Thư viện này được mô tả trong tài liệu: “Atmel AVR201: Using the AVR Hardware Multiplier”. Mã nguồn được chứa trong file avr201.asm.

Trong công thức $V_{IN} = \frac{ADCVal * V_{REF}}{1024}$, sau khi nhân ta còn phải thực hiện phép chia cho 1024.

Phép chia này thực hiện đơn giản bằng cách dịch số bị chia qua phải 10 bit. Phép chia này là phép chia nguyên, kết quả là 1 số nguyên và bỏ qua số dư, không có phần thập phân.

Để có thể hiển thị kết quả có phần thập phân, ta lựa chọn số lượng số sau dấu . thập phân và nhân số ADCVal * V_{REF} cho lũy thừa của 10 trước khi tiến hành phép chia.

Ví dụ ta muốn hiển thị 4 số sau dấu chấm thập phân, ta phải tính giá trị hiển thị theo công thức:

$$V_{IN} * 10000 = \frac{10000 * ADCVal * V_{REF}}{1024}$$

Và hiển thị con số này, với 4 số cuối là số sau dấu.

Nếu ta dùng V_{REF} = 2.56V, công thức trên trở thành:

$$V_{IN} * 10000 = \frac{10000 * ADCVal * 2.56}{1024} = \frac{100 * ADCVal * 256}{1024} = 25 * ADCVal.$$

Để dễ dàng kiểm tra kết quả, ta có thể kết nối tín hiệu điện áp này vào khối TEST STATION và dùng VOM hoặc oscilloscope để đo giá trị thực tế.

Đoạn chương trình sau thực hiện việc khởi động và đọc ADC kênh 0

```
.include "m324padef.inc" ; Include Atmega324pa definitions

call init_adc
start:
    call read_adc_16bit
    rjmp start
;init the ADC with reference voltage to AVCC, select ADC channel 0
;set ADC clock to 125Khz with CPU clock 1Mhz
init_adc:
    ldi r16, (1<<REFS0) ; set reference voltage to AVCC, ADC channel 0
    sts ADMUX, r16 ; store the value in ADMUX register
    ldi r16, (1<<ADEN) | (1<<ADPS1)|(1<<ADPS0) ; set ADC prescaler to 8, enable
ADC
    sts ADCSRA, r16 ; write to ADCSRA register
    nop
    ret
;start a single conversion
;read ADC and store H-L byte to r17-r16
read_adc_16bit:
    push r18

    lds r18, ADCSRA
    ori r18, (1<<ADSC)
    sts ADCSRA, r18

    read_adc_16bit_wait:
        lds r18, ADCSRA
        andi r18, (1<<ADSC)
```

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

```
cpi          r18, (1<<ADSC)
breq      read_adc_16bit_wait
lds       r16, ADCL           ; read ADCL first
lds       r17, ADCH           ; read ADCH second

pop       r18
ret
```

Ví dụ 18 Giao tiếp ADC ở chế độ single end

11.2.2.2 LẬP TRÌNH ADC Ở CHẾ ĐỘ VI SAI

Ở chế độ vi sai, ADC được sử dụng để đo điện áp giữa 2 tín hiệu. Các tín hiệu điện áp được đưa vào cặp các ngõ vào P (dương) và N (âm), từ đó hiệu điện thế được khuếch đại và đưa vào bộ ADC. Các cặp tín hiệu và độ lợi được chọn bằng cách cấu hình thanh ghi ACDMUX, với các tham số được cho ở bảng 25-4, trang 319, tài liệu: Atmel-42743B-ATmega324P/V_Datasheet_Complete-08/2016.

Lưu ý rằng ở chế độ vi sai, điện áp tham chiếu phải lớn hơn 2V và nhỏ hơn AV_{CC} – 0.5. Vì vậy, ta phải dùng điện áp 2.56V onchip hoặc chỉnh nguồn tạo điện áp tham chiếu bên ngoài và đưa vào AV_{REF} cho phù hợp.

Lưu ý là với chế độ vi sai, kết quả sẽ là số có dấu ở dạng bù 2

$$ADCVal = \frac{(V_{POS} - V_{NEG}) * GAIN * 512}{V_{REF}}$$

Đoạn chương trình sau đo sai lệch điện áp giữa 2 chân ADC1 và ADC0

```
.include "m324padef.inc" ; Include Atmega324pa definitions

call  init_adc
start:
    call   read_adc_16bit
    rjmp  start

init_adc:
    ldi   r16, (1<<REFS0)|(1<<REFS1) | 0b00001001 ; set reference voltage to
2.56, , ADC1 (P), ADC0 (N) GAIN=10
    sts   ADMUX, r16        ; store the value in ADMUX register
    ldi   r16, (1<<ADEN) | (1<<ADPS1)|(1<<ADPS0) ; set ADC prescaler to 8, enable
ADC
    sts   ADCSRA, r16       ; write to ADCSRA register
    nop
    ret

;start a single conversion
;read ADC and store H-L byte to r17-r16
read_adc_16bit:
    push  r18

    lds  r18, ADCSRA
    ori  r18, (1<<ADSC)
    sts  ADCSRA, r18

read_adc_16bit_wait:
    lds  r18, ADCSRA
```

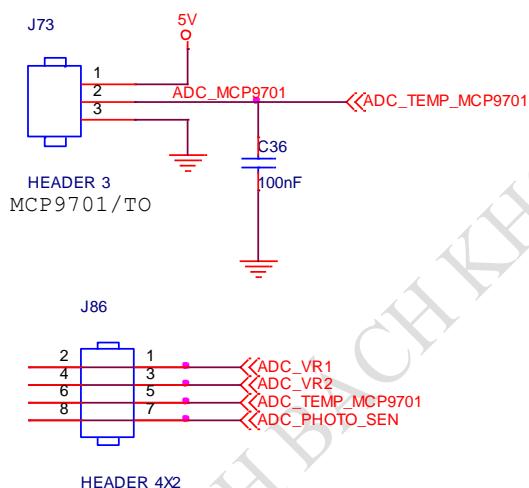
```

andi    r18, (1<<ADSC)
cpi     r18, (1<<ADSC)
breq   read_adc_16bit_wait
lds    r16, ADCL           ; read ADCL first
lds    r17, ADCH           ; read ADCH second
pop    r18
ret

```

11.3 CẢM BIẾN NHIỆT ĐỘ MCP9701

11.3.1 THIẾT KẾ PHẦN CỨNG



Hình 37 Sơ đồ khối cảm biến nhiệt độ analog

Cảm biến nhiệt độ MCP9701 của Microchip là 1 cảm biến có ngõ ra điện áp. Ngõ ra này được kết nối đến header J86.

11.3.2 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

Để đo nhiệt độ, ta kết nối tín hiệu ADC_TEMP_ADC9701 trên header J86 vào 1 kênh ngõ vào analog của vi điều khiển và sử dụng ADC để đo điện áp này, từ đó tính ra nhiệt độ.

Để dễ dàng kiểm tra kết quả, ta có thể kết nối tín hiệu điện áp này vào khối TEST STATION và dùng VOM hoặc oscilloscope để đo giá trị thực tế.

Ngõ ra của MCP9701 được tính theo công thức sau:

$$V_{out} = T_c * T_a + V_{0c}$$

Với:

Ta: nhiệt độ môi trường

Tc: hệ số nhiệt, đối với MCP9701, Tc=19.53mV. Vậy, cứ mỗi 1 độ C, ngõ ra tăng 19.53

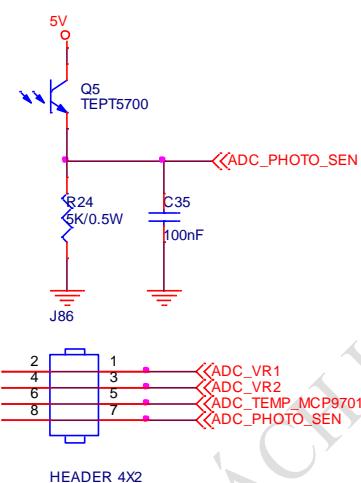
mV

V_{0c}: điện áp tại 0 độ C. Với MCP9701, V_{0c} = 400 mV.

Lưu ý rằng giá trị 400 mV là giá trị thông thường. Trong các ứng dụng thực tế, ta phải tìm ra giá trị chính xác trong quá trình cân chỉnh, thường làm bằng cách nhúng cảm biến vào nước đá đang tan và đo giá trị điện áp ngõ ra.

11.4 CẢM BIẾN ÁNH SÁNG

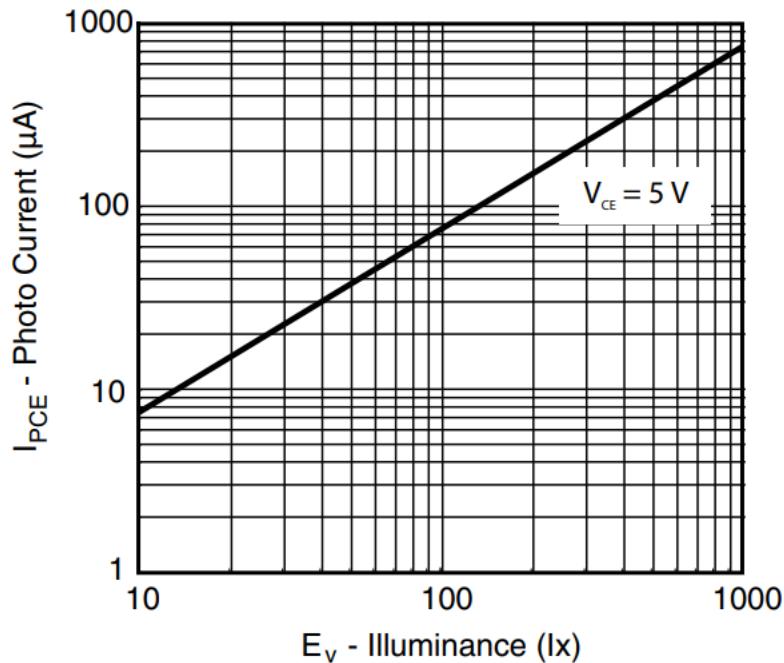
11.4.1 Thiết kế phần cứng



Hình 38 Thiết kế khối cảm biến ánh sáng

Để đo độ sáng, ta sử dụng một transistor quang (photo transistor) TEPT5700. Cảm biến này có dòng điện IC phụ thuộc vào ánh sáng nó nhận được. Dòng điện này được đưa qua điện trở R24 để chuyển đổi thành điện áp. Tín hiệu điện áp này được kết nối vào header J86.

11.4.2 Kết nối phần cứng và lập trình



Hình 39 Đặc tuyến ngõ ra của TEPT5700

Với R24 = 5Kohm, điện áp ngõ ra của cảm biến sẽ là

$$V_{out} = 5K * I_{CPE} [1]$$

Khi cường độ ánh sáng tăng lên thì dòng điện Icpe sẽ tăng lên, làm cho điện áp ra tăng dần về 5V. Như trên Hình 39, khi độ sáng là 900 lx thì cường độ dòng điện ngõ ra là 800uA. Khi đó $V_{out} = 4V$.

Khi cường độ sáng tăng lên nữa, Vce của TEPT5700 giảm về Vceo (điện áp Vce bão hòa), cảm biến sẽ đi về trạng thái bão hòa nên không còn hoạt động đúng.

Để chuyển đổi từ giá trị điện áp đo được sang cường độ ánh sáng, ta sử dụng đặc tuyến ở Hình 39. Đây là đặc tuyến dạng log-log, khi được biểu thị bằng 1 đường tuyến tính nghĩa là quan hệ giữa trực tung và trực hoành sẽ có dạng:

$$n \log I_{CPE} + \log k = \log E_v [2]$$

Từ đặc tuyến ở trên, ta có các điểm tham chiếu sau:

$$E_v = 10, I_{CPE} = 7$$

$$E_v = 900, I_{CPE} = 700$$

Từ đó ta suy ra hàm tương quan giữa I_{CPE} và E_v và sử dụng hàm này để tính ra I_{CPE} khi đo được E_v .

Để đo ánh sáng, ta kết nối tín hiệu ADC_PHOTO_SEN trên header J86 vào 1 kênh ngõ vào analog của vi điều khiển và sử dụng ADC để đo điện áp này, từ đó tính ra độ sáng.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Để dễ dàng kiểm tra kết quả, ta có thể kết nối tín hiệu điện áp này vào khối TEST STATION và dùng VOM hoặc oscilloscope để đo giá trị thực tế.

Vì tương quan giữa các đại lượng là 1 hàm mũ, nên rất khó hiện thực các phép tính bằng assembly. Cho dù sử dụng các thư viện C thì do giới hạn phần cứng tính toán, các hàm tính toán cũng chạy rất chậm.

Để thực hiện việc tính toán ra độ sáng từ giá trị ADC, ta có thể sử dụng phương pháp tra bảng. Với ADC 10bit, ta sẽ lập một bảng gồm có 1024 độ sáng ứng với giá trị đọc ADC từ 0 đến 1023. Với độ sáng trong tầm từ 0-1000lx, mỗi giá trị trong bảng sẽ cần 2 byte, do đó ta sẽ cần 2KB cho bảng này.

11.5 CẢM BIẾN NHIỆT ĐỘ DS18B20

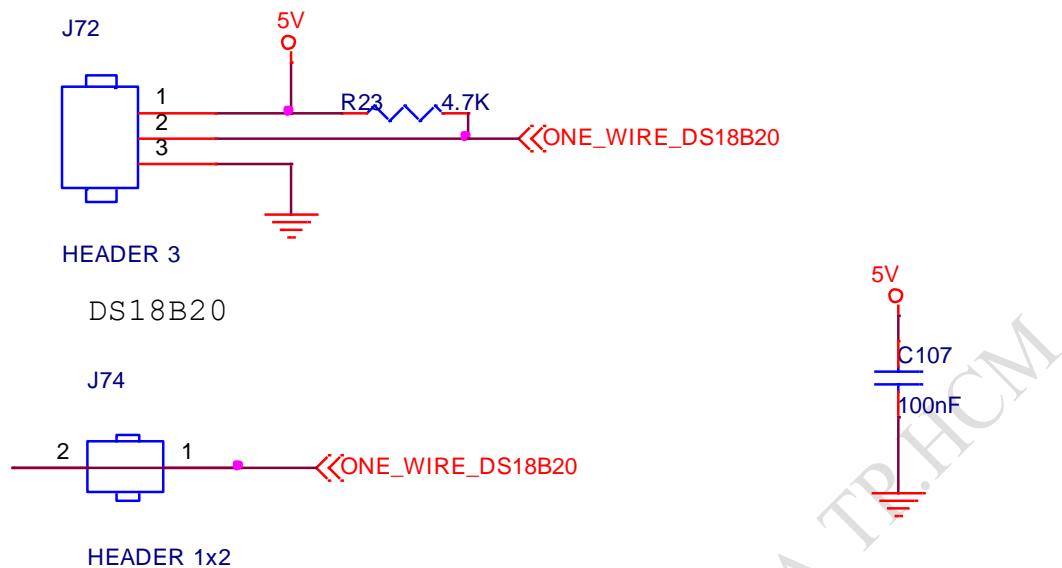
DS18S20 là vi mạch số đo độ C 9-bit với chức năng cảnh báo khi nhiệt độ vượt quá một ngưỡng trên hoặc dưới lập trình được. Vi mạch thông tin trên bus 1 dây (1-Wire bus), tức là chỉ cần 1 chân port để truyền tin giữa cảm biến và MCU. Tầm đo của vi mạch trong khoảng -55°C đến +125°C với độ chính xác tối đa là 0.5°C. Bên cạnh đó, DS18S20 có thể được cấp nguồn trực tiếp thông qua đường dữ liệu.

Mỗi DS18S20 có chứa một mã nối tiếp 64-bit duy nhất, điều đó cho phép nhiều vi mạch có thể cùng hoạt động trên cùng bus 1 dây. DS18S20 gồm 3 chân: GND, DQ, và V_{DD}, trong đó DQ là chân vào và ra của dữ liệu.

Giao thức bus 1 dây (1-Wire) là một giao thức được Dallas định nghĩa. Đường điều khiển cần có điện trở kéo lên vì tất cả các vi mạch đều được kết nối đến bus thông qua 1 cổng 3 trạng thái hoặc cực máng hở (chân DQ). Trong hệ thống bus này, MCU (master) nhận dạng và định địa chỉ các vi mạch trên bus bằng mã địa chỉ 64-bit. Vì mỗi vi mạch có một mã nhận dạng 64-bit duy nhất nên gần như là không giới hạn số vi mạch kết nối trên bus.

11.5.1 THIẾT KẾ PHẦN CỨNG

Trên kit thí nghiệm, J72 được thiết kế để kết nối với cảm biến DS18B20. Ngõ ra tín hiệu được kết nối với J74.



Hình 40 Sơ đồ thiết kế khối DS18B20

11.5.2 KẾT NỐI VÀ LẬP TRÌNH GIAO TIẾP

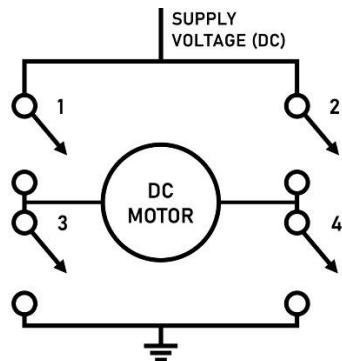
Để giao tiếp với cảm biến, ta kết nối cảm biến DS18B20 với J12, và kết nối J74 đến 1 chân port của vi điều khiển.

Cách đọc giá trị từ DS18B20, sinh viên tham khảo tài liệu datasheet của cảm biến này. Các cách giao tiếp AVR với cảm biến này, có thể tham khảo tài liệu: “AVR318: Dallas 1-Wire Master on tinyAVR and megaAVR”

CHƯƠNG 12 ĐỘNG CƠ DC

12.1 TỔNG QUAN LÝ THUYẾT

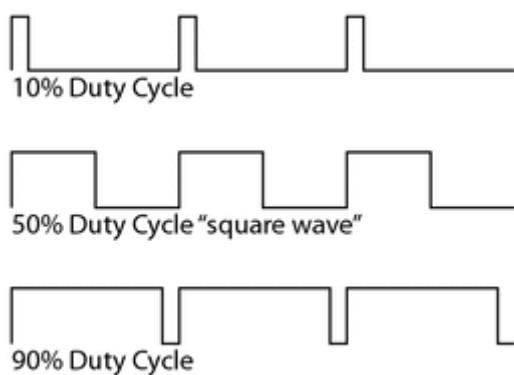
Động cơ DC là đối tượng thường gặp trong việc điều khiển. Động cơ DC được điều khiển bằng cách cấp một dòng điện một chiều chạy qua stator. Chiều quay và tốc độ được thay đổi bằng cách thay đổi chiều dòng điện và cường độ dòng điện. Ngoài ra, động cơ DC còn có thể được điều khiển momen xoắn (torque) khi cần làm việc với các tải thay đổi.



Hình 41 Cầu H

Phương pháp điều khiển thông dụng nhất là sử dụng một cầu H. Các khóa sẽ đóng/mở để cho thay đổi chiều dòng điện chạy qua động cơ, từ đó thay đổi chiều quay. Nếu 2 khóa 1 và 4 đóng, động cơ quay theo chiều thuận, nếu 2 khóa 2 và 3 đóng, động cơ quay theo chiều nghịch. Khi các khóa mở, không có dòng điện chạy qua thì động cơ sẽ giảm tốc và dừng lại.

Để thay đổi tốc độ, các khóa không đóng liên tục mà đóng/mở sử dụng một xung PWM với tần số cố định và chu kỳ làm việc thay đổi, từ đó thay đổi cường độ dòng điện trung bình chạy qua động cơ. Động cơ được dừng nhanh (thắng) bằng cách đóng đồng thời khóa 1-2 hoặc 3-4.

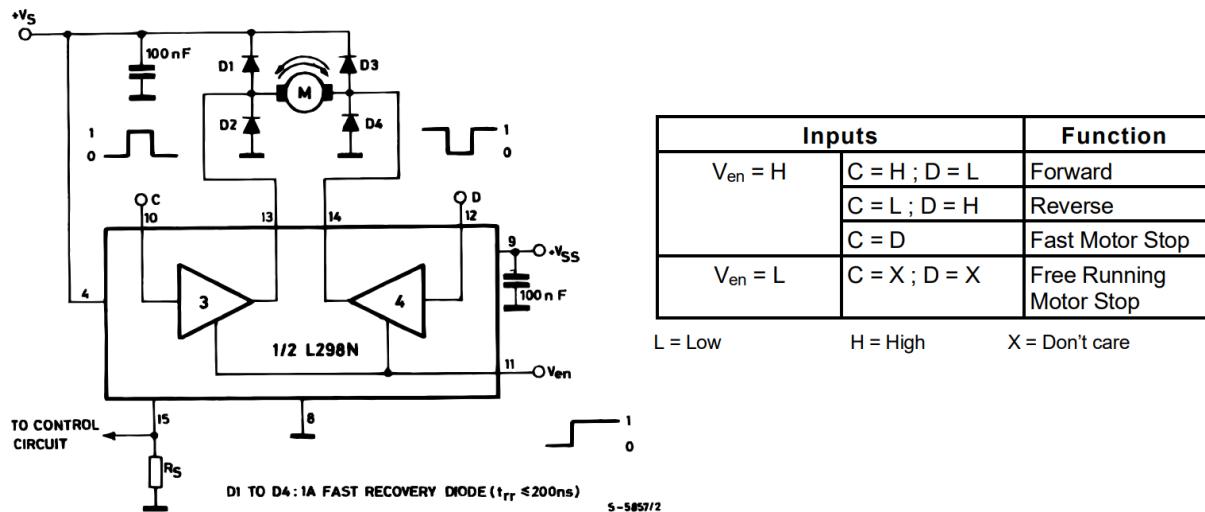


Hình 42 Điều khiển tốc độ động cơ DC dùng PWM

Trên kit thí nghiệm sử dụng một module cầu H L298N.

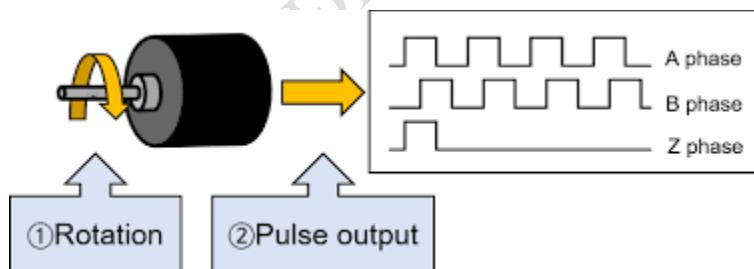
HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Các khóa được tích hợp bên trong module, điều khiển bằng cách đưa các tín hiệu thích hợp vào các chân điều khiển EN, C, D. IC l298 được tích hợp 2 cầu H.



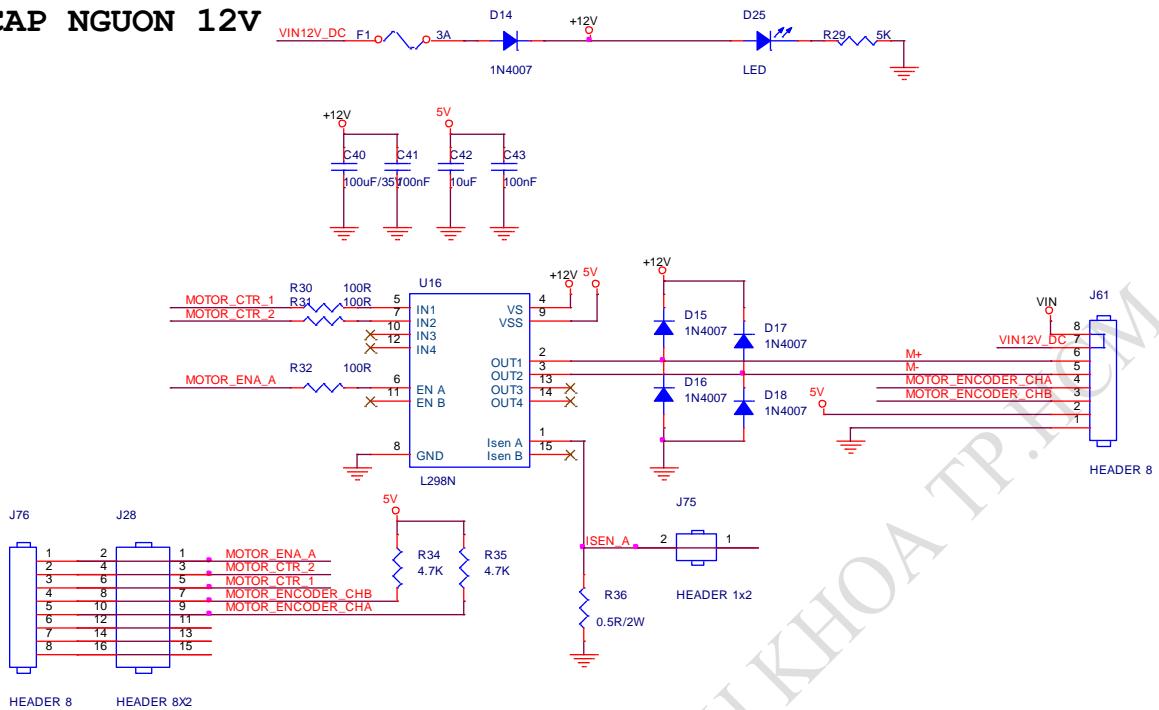
Hình 43 Nguyên lý hoạt động của cầu H L298N

Để đo tốc độ thực tế của động cơ, một encoder được gắn vào trục động cơ. Tùy theo mỗi loại encoder mà ứng với một vòng quay của động cơ sẽ có 1 số lượng xung phát ra. Các encoder còn được thiết kế xuất ra 2 xung có pha lệch nhau để xác định chiều quay và vị trí của động cơ. Ngoài ra, một số encoder còn có xung Z để báo động cơ đi qua vị trí zero trong trường hợp cần đếm số vòng quay.



12.2 THIẾT KẾ PHẦN CỨNG

CẤP NGUỒN 12V



Hình 44 Sơ đồ thiết kế khối điều khiển động cơ

Thiết kế sử dụng IC L298, dùng 1 cầu H. Các tín hiệu điều khiển đóng/mở khóa đưa MOTOR_CTRL_1 và MOTOR_CTRL_2 đưa vào ngõ vào IN1 và IN2. Tín hiệu MOTOR_ENABLE cho phép cầu hoạt động. Các ngõ ra OUT1-OUT2 gắn vào hai đầu động cơ.

Dòng điện chạy qua động cơ sẽ đi qua điện trở R36, gây ra điện áp trên tín hiệu ISEN_A. Ta kết nối tín hiệu này vào một kênh ADC để đo dòng điện chạy qua động cơ.

Động cơ có tích hợp 2 kênh encoder xung A và B, được kết nối ra header J28. Ta sử dụng các tín hiệu này để tính tốc độ và chiều quay thực tế của động cơ.

12.3 KẾT NỐI PHẦN CỨNG VÀ LẬP TRÌNH

Kết nối tín hiệu MOTOR_CTRL1 và MOTOR_CTRL2 vào 2 chân port để điều khiển chiều. Kết nối tín hiệu MOTOR_ENABLE vào một chân ngõ ra PWM để điều khiển tốc độ động cơ. Đưa 1 tín hiệu encoder vào ngõ vào đếm xung của 1 timer để đếm số xung. Có thể đưa 2 tín hiệu này vào 2 chân port khác của AVR để xác định chiều quay.

12.3.1 Điều khiển động cơ

Động cơ được điều khiển bằng các tín hiệu MOTOR_ENABLE, MOTOR_CTRL1 và MOTOR_CTRL2. Các chế độ hoạt động được mô tả ở Hình 43.

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Nếu MOTOR_ENABLE bằng 0 thì động cơ sẽ không được cấp nguồn và sẽ quay chậm dần theo quán tính.

Để thăng động cơ, ta cho MOTOR_ENABLE = 1, hai tín hiệu MOTOR_CTRL1, MOTOR_CTRL2 cùng bằng 1 hoặc bằng 0.

Để động cơ chạy thuận thì tín hiệu MOTOR_ENABLE và MOTOR_CTRL1 bằng 1, MOTOR_CTRL2 = 0. Để động cơ chạy thuận thì tín hiệu MOTOR_ENABLE và MOTOR_CTRL2 bằng 1, MOTOR_CTRL2 = 0.

Để điều khiển tốc độ, ta đưa xung PWM vào tín hiệu MOTOR_ENABLE và cho một trong 2 tín hiệu thuận/ngược tích cực.

Tùy mỗi loại động cơ mà sẽ cần tần số xung PWM khác nhau. Đối với động cơ dùng trong bài thí nghiệm, xung PWM với tần số 1Khz là phù hợp.

Tín hiệu PWM có thể tạo ra sử dụng timer. Đối với AVR, nếu ta dùng timer 0 để tạo xung PWM và muốn điều chỉnh được tần số thì phải sử dụng chế độ Fast PWM ở mode 7 hoặc Phase Correct ở Mode 5.

12.3.2 Đo tốc độ động cơ

Động cơ sử dụng trong thí nghiệm có đặc tính sau:

Tốc độ quay	333 rpm
Điện áp	12 V dc
Loại	Có encoder
Đường kính trục	6mm
Chiều dài động cơ	47mm
Đường kính động cơ	37mm
Tỉ số truyền	1:30
Xung / Vòng	11 xung
Series	GB37

HƯỚNG DẪN THÍ NGHIỆM VI XỬ LÝ

Động cơ này đã có hộp giảm tốc với tỉ số truyền là 1:30. Encoder phát ra 11 xung/vòng. Vì vậy, khi trục động cơ quay 1 vòng thì ta sẽ có 330 xung, tương ứng với 30 vòng quay của đĩa encoder.

Tốc độ 333 rpm (số vòng/phút) trong datasheet là tốc độ quay của trục động cơ đã qua giảm tốc.

Để đo tốc độ động cơ, có hai cách thông dụng:

1. Đếm số xung từ encoder trong vòng 1 s, từ đó tính ra số vòng quay trong 1s.
2. Đếm thời gian cần để quay đủ 1 vòng, từ đó tính ra tốc độ.

Với phương pháp 1, tín hiệu từ encoder được đưa vào ngõ vào của 1 timer. Một timer khác được sử dụng để đếm thời gian 1 s. Sau khi hết 1 s này thì ta đọc số đếm của timer đếm xung. Giả sử ta đếm được 330 xung thì có nghĩa là tốc độ trong giây vừa rồi là 1v/s hay 60rpm (revolutions per minute).

Với phương pháp 2, timer đếm xung được cấu hình sao cho khi số đếm đạt 330 xung thì báo ngắt. Timer còn lại dùng để đếm thời gian từ khi số xung bằng 0 cho đến khi bằng 330. Từ thời gian này ta có thể tính ra tốc độ motor.

Thay vì sử dụng ngõ vào timer để đếm xung, ta có thể đưa tín hiệu từ encoder vào chân ngắt ngoài. Cứ mỗi khi có một xung cạnh xuống (hoặc cạnh lên) thì ta tăng số đếm lên 1. Với phương pháp này ta sẽ không phải dùng 1 timer để đếm xung, và có thể dùng timer này cho một tác vụ khác.